Master's Thesis
in Data Science

From Node Embeddings to Graph Generation

Josifovska Angela

Supervisors: Prof. Dr. Stephan Günnemann
Oleksandr Shchur
Technical University of Munich

Prof. Dr. Matthias Schubert
Ludwig Maximilian University of Munich

Submission date: 20.12.2019

Declaration of Authorship

I hereby declare that the thesis submitted is my own unaided work. All direct or indirect sources used are acknowledged as references.
This paper was not previously presented to another examination board and has not been published.

Munich, 20.12.2019

.....
Josifovska Angela

Abstract

Node embeddings are a popular choice for graph representation in machine learning and are widely used for tasks like node classification and link prediction. However, it is not clear which graph properties they can capture and how well. In this thesis we give a fresh look on node embeddings, by examining their capabilities and limitations in (i) capturing graph properties and (ii) generating graphs. Our work led to surprising insights about the sensitivity of the embeddings to regularization and the (in)ability to learn meaningful embeddings from sparse graphs. Furthermore, we proposed a framework for graph generation using node embeddings, by performing density estimation on the embeddings. We used state-of-the-art methods for density estimation and confirmed their flexibility in estimating complex densities. Our work shows that the proposed approach for graph generation is promising, however, it faces considerable challenges that need to be addressed for a successful realization.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Graphs are ubiquitous data structures used in wide range of real-world applications, from social networks and recommendation systems, to knowledge bases and systems for navigation. The applicability of graphs in various fields has motivated the development of machine learning models for graphs for solving tasks like recommending friends on social networks [11] or targeting users in marketing [5].

Node embeddings are a popular choice for graph representation in machine learning and are widely used for tasks like node classification and link prediction. However, little is know about their ability to capture the properties of the graph. Since graphs are complex objects and we cannot easily perceive them like images or text, we use graph statistics to quantitatively assess how well the embeddings capture the graph properties.

Graph generation has applications in fields like biology and social sciences, but most importantly it provides us with better understanding of the structure and behaviour of graphs. One of the earliest works in graph generation is the Erdős–Rényi (ER) random graph model [8], which samples each edge independently with a fixed probability. Other traditional graph generative models aim to capture power law degree distribution [2] or assume certain community structure [13, 15]. Each of these models is "specialized" in modeling a certain type of graph and is not flexible enough to capture all graph properties. The recent advancements in deep learning have enabled the development of deep generative models for graphs, which are more flexible than the traditional models. However, many of them have efficiency and scalability issues or cannot capture all graph properties.

In this thesis we give a fresh look on node embeddings by closely examining their properties and limitations. Specifically, our goal is to provide answers to the following two research questions:

1. **Which graph properties can we capture with node embeddings?**
2. **Can we use node embeddings for graph generation?**

In Chapter 4 we focus on answering the first question. We use simple embedding models and we inspect their ability of capturing the properties of various types of graphs. Our approach in examining which properties we can capture involves (i) training node embbedings from the original graph, (ii) reconstructing a graph from the trained node embeddings and (iii) comparing the properties of the original and reconstructed graphs. In the first part of this chapter, we perform experiments on simple synthetic graphs which allow us to examine each property separately. In the second part, we experiment with real graphs which are often characterized with a combination of these properties. Our work has resulted in several interesting insights and a better understanding of the capabilities of simple embedding approaches.

Chapter 5 is devoted to answering the second question. Our approach for generating graphs from node embeddings involves performing density estimation on the trained embeddings and sampling from the estimated density. An important aspect of this approach is the choice of a density estimation method that is flexible enough to model the complex shape of the embeddings. We used state-of-the-art methods for density estimation and confirmed their flexibility in estimating complex densities. Our work shows that even though this approach seems promising, there are considerable challenges that need to be resolved in order to have a useful generative model.

Chapter 2

Related work

Traditional approaches. One of the earliest works in graph generation is the Erdős–Rényi (ER) random graph model [8], which generates graphs by sampling each edge independently with a fixed edge probability. Even though it serves as an important mathematical foundation for studying graphs, the ER model fails to capture some of the properties of real graphs, such as the heavy-tailed degree distribution. Similarly, other traditional models [2, 13, 15, 33] are specialized in capturing just a few graph properties, but fail at capturing others.

Deep generative models for graphs. The recent advancements in deep learning have motivated the development of powerful graph generative models based on neural networks. Some deep generative models are generating the edges independently given the latent features. Such models are usually efficient and can be paralelized, however, the independence assumption comes with a risk that they might not be able to capture the complex graph structure. An example of these models is Variational Graph Auto-Encoder (VGAE) [19], which uses Graph Convolutional Network (GCN) [18] as an encoder and a simple dot product as a decoder. Other deep generative models are generating the graphs using an iterative decoding procedure [10], while [22] use normalizing flows and reversible GNNs.

Auto-regressive deep generative models. Some of the most successful deep generative models for graphs use an auto-regressive process to generate the graphs. These models generate nodes in a sequential manner and connect each new node to the existing partial graph [20]. This approach allows them to model the complex dependencies between generated edges, which is challenging for generative models relying on the i.i.d. assumption. However, many auto-regressive models have considerable limitations. For instance, [20] does

not scale well and is more suitable for generation of small graphs. GraphRNN [34] is an auto-regressive model that uses RNN for the sequential generation and achieves better scalability, but suffers from the problem of long-term dependencies in large graphs. Handling permutation-invariance is another serious challenge, as it requires marginalization over all possible node orderings, which is infeasible for large graphs. GraphRNN solves this by using random BFS node orderings, which is efficient, but still suboptimal. Recently, [21] proposed an efficient and scalable auto-regressive model that uses GNNs with attention and canonical node orderings.

Chapter 3

Preliminaries

3.1 Graph and graph properties

A graph or network $G = (V, E)$ is a set of objects V (called nodes or vertices) connected with a set of links (or edges) $E \subseteq V \times V$ [3]. We distinguish between directed and undirected graphs, depending on whether the edges have a direction or not. Weighted graph is a graph in which each edge has a weight or cost, while the edges in unweighted graphs do not have any. If a graph allows multiple edges between two nodes we call it multigraph and otherwise we have a simple graph. In this thesis, we deal with simple graphs that are undirected and unweighted, and which are fully represented by a binary symmetric adjacency matrix $\boldsymbol{A} \in \{0, 1\}^{N \times N}$. The entry $A_{ij} = A_{ji} = 1$ means that nodes i and j are connected with an edge, while a value of zero indicates that they are not connected to each other.

As graphs are complex objects and we as humans cannot easily perceive them like images or text, we use certain graph statistics to understand the important characteristics of a graph and to be able to assess the similarity between graphs. The **graph size** N is defined as the total number of nodes in the graph, while the **graph density** $\rho = \frac{2|E|}{N(N-1)}$ denotes the ratio between the actual number of edges and the maximum number of possible edges. Real networks are usually very sparse and their number of edges is much smaller than the number of possible edges.

The **degree** of a node defines the number of connections of that node to other nodes. To summarize this property for all nodes of the graph, we are usually interested in the minimum, maximum or average degree of the graph, or for a better overview, the whole **degree distribution**. The degree distribution p_k gives the probability that a randomly selected node has a degree k.

Another graph property is the **distribution of the shortest path lengths** between the nodes in the graph. Sometimes, instead of the whole distribution, one is interested in the maximum of the shortest path lengths, called **diameter**. Low shortest path lengths (small diameter) can indicate small-world graphs, which is an interesting phenomenon of real networks where the nodes are just few connections away from any other node [33].

Small-world graphs are also characterized by a high **average clustering coefficient**, which is defined as the average of the local clustering coefficients of all nodes. A **local clustering coefficient** of a node quantifies how much its neighbour nodes tend to connect to each other, i.e. how densely connected is the neighbourhood of that node.

A triangle is a triple of nodes that are all connected between each other. The **number of triangles** is used for calculating the **global clustering coefficient** which is proportional to the ratio between the number of triangles and the number of all triplets (three nodes connected by at least two edges). The number of triangles and the global clustering coefficient are properties that show the global tendency of the nodes to form clusters. Social networks for instance tend to have high number of triangles because friends of friends are typically friends themselves [32].

3.2 Generative models for graphs

In order to study the properties and behaviour of real networks, as well as to generate synthetic graphs with specific predefined properties, various graph generating models have been developed.

3.2.1 Erdős–Rényi (ER) model

One of the oldest graph generating models is the **Erdős–Rényi (ER) random graph model** [8]. The Erdős–Rényi model generates random graphs, i.e. graphs in which connections between the nodes occur at random. There are two versions of the model. The first has two parameters: N, the number of nodes and L, the number of edges and it generates a graph by randomly selecting L pairs of nodes from the total of $\frac{N(N-1)}{2}$ node pairs and connects them.

The second version takes two parameters: N, the number of nodes and p, the probability of having an edge between two arbitrary nodes. Unlike the

first version which assumes fixed number of edges, this version of the model has fixed probability p, hence the number of edges for the generated graphs with same size can vary between samples. The probability p is the expected density of the graph, meaning that low p will lead to sparse graphs and high p to more dense graphs. In the rest of the thesis, this version of the ER model will be used.

The degree distribution of a random network follows a binomial distribution:

$$p_k = \binom{N-1}{k} p^k (1-p)^{(N-1-k)} \tag{3.1}$$

That is, the probability p_k of an arbitrary node having degree k can be interpreted as the probability of getting k successes (neighbours) in $N-1$ trials (possible candidates for neighbours are the rest $N-1$ nodes). The expected degree $\boldsymbol{E}[k]$ is therefore equal to $p(N-1)$. If N is much larger than the expected degree $\boldsymbol{E}[k]$, the degree distribution can be approximated with Poisson distribution:

$$p_k = \frac{\lambda^k e^{-\lambda}}{k!}, \tag{3.2}$$

where $\lambda = \boldsymbol{E}[k]$.

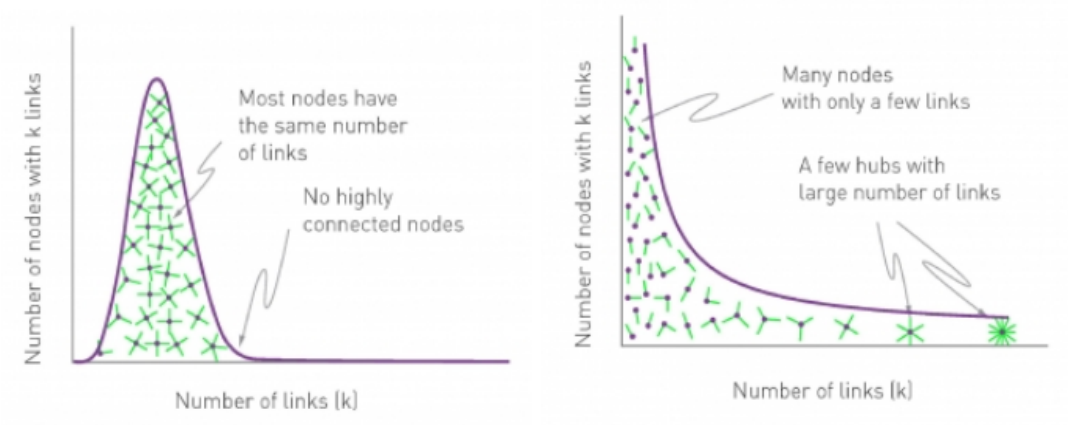
Even though it serves as an important mathematical foundation for studying graphs, the random graph model cannot capture the behaviour and properties of the real graphs well. In fact, the graphs produced by this model lack community structure and have homogeneous degree distribution. On the other hand, real networks often form communities and contain few nodes which have very high degrees, while the others have significantly lower connections.

3.2.2 Barabási–Albert (BA) model

The **Barabási–Albert (BA) model** [2] overcomes one of the limitations of ER graph - the ability to generate graphs with realistic degree distribution. Many existing networks such as the WWW, paper citations, protein interactions and social networks have been found to be so called scale-free networks [3], which means that their degrees follow a power law distribution:

$$p_k = ck^{-\alpha}, \tag{3.3}$$

where c is a multiplication constant that shifts the degree distribution and α is the degree exponent. From Equation 3.3, it follows that $\log p_k$ and $\log k$ have a linear relationship, so if we plot the degree distribution on logarithmic scales, it will look like a straight line with a slope controlled by α.



(a) Poisson (b) power law

Figure 3.1: Comparison between Poisson and power law degree distributions [3]

Figure 3.1 shows the difference between the degree distributions of a random graph and a scale-free graph. In a random graph the majority of the nodes have comparable degrees and there are no hubs (nodes with very high degree), whereas in scale-free graphs there are a few hubs and most of the nodes have very low degrees. For instance, in social networks, there are small number of users that are quite popular and have acquired many followers and connections, while the most of the "regular" users have smaller circle of online friends.

The Barabási-Albert model generates scale-free graphs by starting with an initial set of m0 nodes, that are arbitrarily connected. Then, in each step, a new node is added to the network and it is connected to m already existing nodes. The probability that the new node will be connected to a particular existing node depends on its degree, i.e. there is a higher probability of connecting with a high degree node. This means that as the network grows, the high degree nodes will attract more and more new nodes, which will lead to few nodes with very high degree and majority of nodes with low degree. It can be shown that the BA model always generates graphs whose degrees follow a power law distribution with degree exponent $\alpha = 3$ [3].

The advantage of BA model in comparison to the ER model is its ability to generate graphs with power law degree distribution. However, the BA model is not flexible enough and it can only produce degrees from power law distribution with a fixed degree exponent $\alpha = 3$.

3.2.3 Degree-corrected Erdős–Rényi (ER-DC) model

The **degree-corrected Erdős–Rényi (ER-DC) model** [15] overcomes the limitation of the BA model and it can generate graphs with power law degree distribution with a flexible degree exponent α. This model takes 3 parameters: the number of nodes N, the edge probability p and the degree sequence $\boldsymbol{\theta}$. The parameters $\boldsymbol{\theta}$ actually denote the expected degree of every node divided by $\sqrt{p}$. We can obtain $\boldsymbol{\theta}$ by sampling a sequence of length N from a power law degree distribution with own choice for the values of α and c. By choosing different values for α and c, the ER-DC model can generate graphs with with different density and varying slope of the power law degree distribution.

The expected value of the adjacency matrix element A_{ij} in ER-DC depends on the expected degrees of the two nodes i and j. The graph is sampled as:

$$A_{ij} \sim Poisson(\theta_i \theta_j p) \tag{3.4}$$

after which the values of the adjacency matrix larger than 1 need to be clipped to 1 as we do not consider multigraphs.

3.2.4 Watts–Strogatz model (WS)

Six degrees of separation is well known theory stating that a person is at most 6 connections away to any other person on the planet. This small world phenomenon is found to be true for many real networks and indicates that the shortest path lengths between nodes is quite small, while at the same time they have high clustering coefficient (which means that the nodes have high tendency to cluster together). For instance, it was observed that social networks like Facebook, are sparse, have low diameter and high clustering coefficient at the same time [31]. This is counter-intuitive because the graph as a whole is sparse, but the graph neighborhoods of users are surprisingly dense; also it seems that users from different social circles are far away from each other, however, the average distance between pairs of users is shown to be surprisingly small.

The ER model cannot capture these small world properties. In fact, random graphs generated by ER model have short paths thanks to the random placement of edges, however, they have low clustering coefficient. On the other hand, lattice graphs have high clustering, but large distances between the nodes (in terms of a social network, each individual would only know its neighbours). The **Watts-Strogatz model** [33] is capable of creating small-world graphs

with both short paths between nodes and high clustering coefficient, by interpolating between a regular lattice and a random graph.

The model places the nodes in a ring and connects each node with its K intermediate neighbours. Then each edge is rewired at random with probability p. As we increase p from 0 to 1, we move from a regular lattice to a random graph, and somewhere in between we can observe graphs with both low shortest path lengths and high clustering.

3.2.5 Stochastic block model (SBM)

The nodes in real networks tend to group together and form communities, yet none of the previously explained models were able to generate graphs with communities. **Stochastic Block Model (SBM)** [13] is a generative model that is able to create graphs with a specific community (or also called block) structure. To generate a graph with N nodes and K clusters, the SBM model first needs a block membership vector $\boldsymbol{z}$ of size N, where the element $z_i \in [K]$ denotes the cluster which node i belongs to. Second, we need a $K \times K$ matrix $\mathbf{B}$ which contains the edge probabilities between each two clusters. If $z_i = k$ and $z_j = t$ (meaning that nodes i and j belong to clusters k and t respectively), then the SBM model creates an edge between i and j with probability $B_{z_i z_j} = B_{kt}$. So, the edges are drawn i.i.d. with a probability that depends on the clusters that the nodes belong to:

$$A_{ij} \sim Bernoulli(B_{z_i z_j}) \tag{3.5}$$

A simple version of SBM is the case where we have only two edge probabilities, one for nodes of the same cluster (within-cluster probability) and one for nodes of different clusters (between-cluster probability):

$$B = \begin{pmatrix} p_w & p_b \\ p_b & p_w \end{pmatrix} \tag{3.6}$$

where p_b and p_w are the between-cluster and within-cluster probabilities, respectively.

Same as the ER model, SBM comes with the limitation of creating graphs with homogeneous degree distribution. Therefore, a degree-corrected version has been developed, that allows for creating graphs with any given degree sequence (including degrees following a power law distribution). The **degree-corrected SBM (SBM-DC)** [15] additionally takes a degree sequence $\boldsymbol{\theta}$ as an input and it calculates the expected value of the adjacency matrix element

A_{ij} as $\theta_i \theta_j B_{z_i z_j}$. The graph is generated by sampling the adjacency matrix from a Poisson distribution:

$$A_{ij} \sim Poisson(\theta_i \theta_j B_{z_i z_j}) \tag{3.7}$$

and as the values of the sampled adjacency matrix can be larger than 1, they need to be clipped to 1 (as we work with simple graphs).

3.2.6 Random Geometric Graph (RGG)

Random Geometric Graph (RGG) [27] is a random spatial graph generated by sampling N points from a uniform distribution in a $[0, 1)^d$ metric space (usually Euclidean). The nodes are then connected if they are close to each other, i.e. if their distance is smaller than a predefined neighbourhood radius r:

$$A_{ij} = \begin{cases} 1, & \text{if } d(i,j) \leq r \\ 0, & \text{otherwise} \end{cases} \tag{3.8}$$

where $d(i, j)$ is the distance between the nodes i and j. As the nodes are connected by proximity, if node A is close (and thus connected) with nodes B and C, then probably B and C are connected as well. Therefore this graph is characterised with high number of triangles that increase with the radius.

3.3 Node embeddings

The idea of node embeddings is to map the original nodes of the graph into points in a low-dimensional vector space $\mathbb{R}^d$. Representation learning approaches [11] treat node embeddings as vector representations which encode some information about the node and its neighbourhood. After learning the embedding vectors, they can be used as inputs to machine learning models for solving tasks like node classification, link prediction and community detection. Some methods learn the embeddings using matrix-factorisation approaches [4], while others like Node2vec [9] optimize the embeddings such that their dot product reflects the similarity of nodes computed with random walks. Other methods create the embeddings by aggregaring the node's attributes and the attributes of its neighbours using neural networks [18].

Some approaches view the node embeddings as unobserved latent features which reflect some characteristics of the nodes [12]. The nodes are likely to be connected if their latent positions are close in the latent space (meaning that they share similar characteristics). Latent space models assume the edges to

be conditionally independent given the latent positions and model the edge probability of two nodes i and j with a function of their latent features $\boldsymbol{z_i}$ and $\boldsymbol{z_j}$:

$$A_{ij} \sim Bernoulli(f(\boldsymbol{z_i}, \boldsymbol{z_j})) \tag{3.9}$$

A way to train these models is by maximizing the likelihood $p(\boldsymbol{A}|\boldsymbol{Z})$ given with 3.9. Having a trained embedding matrix $\boldsymbol{Z}$, we can reconstruct a graph from it by sampling the edges i.i.d. from 3.9. In this way, latent space models allow for probabilistic inference and are suitable for graph generation.

There are different options for the choice of f in 3.9, which should represent some distance between the latent positions. In Random dot product graphs [35], f is considered to be a function of the dot product of the two latent features. More specifically, [25, 17] use logistic sigmoid function to map the scores from the dot product into edge probabilities. In our work we use a version of the **sigmoid model** containing a trainable bias for the dot product:

$$A_{ij} \sim Bernoulli(\sigma(\boldsymbol{z_i}\boldsymbol{z_j}^T) + b)), \tag{3.10}$$

where $\sigma(x) = (1 + \exp(-x))^{-1}$ is the logistic sigmoid function and b is the bias. Another model that uses the dot product of the embedding vectors is the **Bernoulli-Poisson model** [36, 30], which defines the edge probability as

$$A_{ij} \sim Bernoulli(1 - \exp(-\boldsymbol{z_i}\boldsymbol{z_j}^T)), \quad \boldsymbol{z_i}, \boldsymbol{z_j} \geq 0 \tag{3.11}$$

Instead of using the dot product, there are models that use the distance between the latent features [12, 22]. A version of a **distance model** is given with

$$A_{ij} \sim Bernoulli(\exp(-\|\boldsymbol{z_i} - \boldsymbol{z_j}\|_2^2), \tag{3.12}$$

3.4 Density Estimation

Density estimation is a fundamental task in machine learning. It refers to the problem of approximating the joint probability density $p(\boldsymbol{x})$ of a set of random variables $\boldsymbol{x}$, from given data observations. By estimating the density $p(\boldsymbol{x})$ we are modeling the generative process behind the observed data, which is useful for tasks like statistical inference, predictions, data generation and causal reasoning.

One of the simplest methods for density estimation is the **histogram**, which works by partitioning the data into distinct bins with equal width [6]. Histograms approximate the density of each bin using the relative frequency of

data points that belong to it. Because of their simplicity, histograms are often used for visualization in low dimensions, however, they are not applicable in higher dimensions as the bins become extremely sparse (as a manifestation of the *curse of dimensionality*). Another widely used density estimation method is **Kernel density estimation**, which approximates the density by smoothing the empirical distribution of the data [6]. Despite being a flexible density estimator, this method also suffers from the curse of dimensionality and does not scale well in high dimensions.

Another way to estimate the density is by using parametric models. Unlike Kernel density estimation, these methods define a statistical model $q_\phi(\boldsymbol{x})$ for the density and aim to estimate its parameters ϕ [26]. A typical choice is the Gaussian distribution:

$$q_\phi(\boldsymbol{x}) = \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}), \tag{3.13}$$

where $\phi = (\boldsymbol{\mu}, \boldsymbol{\Sigma})$ are the parameters. The issue with this simple model is that it is not flexible enough to model complex densities. To achieve more powerful density estimation, a mixture of models is often used. A Gaussian Mixture Model (GMM) [24] is a combination of multiple simple Gaussian components:

$$q_\phi(\boldsymbol{x}) = \sum_{k=1}^{K} \pi_k \, \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu_k}, \boldsymbol{\Sigma_k}), \text{ where } \sum_{k=1}^{K} \pi_k = 1 \text{ and } \pi_k \geq 0 \tag{3.14}$$

The combination of multiple models makes GMM a good density estimator. However, to estimate complex densities, it usually needs a large number of components, which is often impractical.

3.4.1 Normalizing Flows

Normalizing Flows [29] are powerful models for density estimation that are flexible enough to approximate complex densities. A normalizing flow is model that transforms a simple distribution (e.g. Uniform or Gaussian) into a complex one, by applying a chain of invertible transformation functions. The chain is called a normalizing flow, because the initial density "flows" through the transformations and the result is a valid probability distribution.

Let $f : \mathbb{R}^d \mapsto \mathbb{R}^d$ be a smooth, invertible function, that maps a random variable $\boldsymbol{z} \sim q(\boldsymbol{z})$ into a new variable $\boldsymbol{y} = f(\boldsymbol{z})$. By applying the change of variable technique, we can obtain the distribution of the resulting variable:

$$q_y(\boldsymbol{y}) = q(\boldsymbol{z}) \left| \det \frac{\partial f^{-1}}{\partial \boldsymbol{y}} \right| = q(\boldsymbol{z}) \left| \det \frac{\partial f}{\partial \boldsymbol{z}} \right|^{-1}, \tag{3.15}$$

where the last equality comes from the inverse function theorem and the property of Jacobians of invertible funcions. In this way, we can approximate arbitrarily complex densities, by successively applying transformations on a random variable $\boldsymbol{z}_0 \sim q_0(\boldsymbol{z}_0)$ and forming a chain of transformation functions f_k:

$$\boldsymbol{z}_K = f_K \circ \cdots \circ f_1(\boldsymbol{z}_0), \tag{3.16}$$

With Equation 3.15 we know the relationship between two consecutive variables in the flow and we can use it to expand the equation for density of the resulting variable $\boldsymbol{z}_K$ step by step backwards until reaching the initial density q_0:

$$\log q_K(\boldsymbol{z}_K) = \log q_0(\boldsymbol{z}_0) - \sum_{k=1}^{K} \log \left| \det \frac{df_k}{d\boldsymbol{z}_{k-1}} \right| \tag{3.17}$$

In this way, the resulting density can be obtained from the initial density q_0 and the determinants of the Jacobians of every transformation f_k. The transformation function f_k needs to fulfill two properties for Equation 3.17 to hold: it needs to be invertible and its Jacobian determinant should be easy to compute.

3.4.2 Block Neural Autoregressive Flow

In order to increase the expressiveness in density estimation, recent works use autoregressive models as normalizing flows. Neural Autoregressive Flow (NAF) [14] is one such method that allows both flexibility and tractability in approximating rich family of distributions. This is achieved by decomposing the transformation function f into autoregressive conditioner c and invertible transformer τ:

$$y_t = f(\boldsymbol{x}_{\mathbf{1:t}}) = \tau(c(\boldsymbol{x}_{\mathbf{1:t-1}}), x_t) \tag{3.18}$$

Each conditioner c_i is a neural network that predicts the pseudo-parameters for the transformer τ_i, which in turn is a monotonic neural network. We can increase the expressively of NAF by using larger transformer networks, however, the conditioner grows quadratically with the size of the transformer network, making the flow inefficient for complex architectures.

Block Neural Autoregressive Flow (B-NAF) [7] overcomes the limitation of NAF, by learning the flow directly, without the use of conditioner networks. In this way, B-NAF requires orders of magnitudes fewer parameters than NAF and can be trained in parallel, while autoregressiveness and monotonicity are still guaranteed. The major drawback of B-NAF (and NAF as well) is that even though the flow is invertible in principle, the inverse is not available in closed form. To be able to both evaluate the density and sample from it we need to access both f and f^{-1}, so the lack of closed-form inverse means that we have to rely on numerical inversion which is unstable and inefficient in high dimensions.

Chapter 4

Capturing graph properties with node embeddings

4.1 Motivation

One of the questions that we aim to answer in this thesis is whether we can generate graphs with node embeddings. Regardless of the approach that we will use for graph generation, the quality of the generated graphs will highly depend on the ability of node embeddings to capture graph properties. Therefore, it is important to inspect the capabilities and limitations of the embedding approaches first, and more specifically to understand what graph properties they can or cannot capture.

In this chapter we experiment with different embedding models and various types of graphs, with the aim of evaluating the ability of the embeddings to capture specific graph properties. We focus on properties such as the degree distribution, clustering coefficient, lengths of shortest paths and number of triangles. First, we train on simple synthetic graphs which allow us to examine each property independently and to see whether the chosen model can reconstruct different types of graphs. Then, we enrich the analysis by using real graphs as well, which exhibit multiple characteristics at once and thus make it more challenging to recover them all with few dimensions. The insights of this analysis will be basis for the next steps in the creation of a graph generative model.

4.2 Experimental setup

Datasets. The experiments in this part were performed on both synthetic and real graphs. For the creation of synthetic graphs we used the following graph generative models (explained in more detail in Section 3.2): ER, ER-DC, BA, WS, SBM, SBM-DC, and RGG model. We experimented with synthetic graphs of size $N = 1000$. For the degree-corrected models we generated degree sequences sampled from power law distribution with $\alpha \in \{1.5, 2, 3\}$ and $c \in \{1, 5, 20\}$. In this case we use the multiplication constant c for controlling the density of the graphs, rather than explicitly using a density parameter p. It also is worth noting that for the degree-corrected graphs, the generated adjacency matrix can contain values larger than one, and we therefore need to clip those values to one. After generating the synthetic graphs, we adjust them to be undirected and without self-loops and we also discard zero-degree nodes.

The real graphs that we used are the citation dataset of machine learning papers Cora-ML [23] and the Political Blogs dataset [1]. We used undirected version of these graphs and only considered the largest connected component (LCC). For Cora-ML, the number of nodes of the LCC is 2,810 and its number of edges is 7,981. The number of nodes and edges of LCC of Political Blogs is 1,222 and 16,714 respectively.

Training node embeddings. For a given graph with adjacency matrix $\boldsymbol{A} \in \{0,1\}^{N \times N}$, we would like to train embedding matrix $\boldsymbol{Z} \in \mathbb{R}^{N \times D}$, such that the likelihood of the graph given the embeddings is maximized. For a given embedding matrix $\boldsymbol{Z}$ we define the edge probability as

$$A_{ij} \sim Bernoulli(f(\boldsymbol{z_i}, \boldsymbol{z_j})), \tag{4.1}$$

where f is the chosen model for the embeddings. We experimented with the sigmoid, Bernouli-Poisson and distance model explained in Section 3.3.

We treat $\boldsymbol{Z}$ as a free variable and optimize it by maximizing the likelihood function $P(\boldsymbol{A}|\boldsymbol{Z})$ given by 4.1. The maximization of the likelihood can be seen as minimizing the negative log likelihood, which is equivalent to using the binary cross-entropy (BCE) loss:

$$\begin{aligned} \mathcal{L}(\boldsymbol{Z}) &= -\sum_{ij} \log p(A_{ij}|f(\boldsymbol{z_i}, \boldsymbol{z_j})) \\ &= -\sum_{ij} A_{ij} \cdot \log f(\boldsymbol{z_i}, \boldsymbol{z_j}) \; - \; \sum_{ij} (1 - A_{ij}) \cdot \log(1 - f(\boldsymbol{z_i}, \boldsymbol{z_j})) \end{aligned} \tag{4.2}$$

The embedding matrix is initialized with values from a standard normal distribution, while the bias for the sigmoid model is initialized as follows:

$$b_{init} = \log \frac{\rho}{(1-\rho)}, \tag{4.3}$$

where ρ is the density of the graph.

The embeddings are trained in 1500 iterations using Adam optimizer with learning rate of 0.1 for the sigmoid and distance model and 0.01 for the Bernoulli-Poisson model. As the datasets we use are not that large, there is no need for mini-batching and the training is done using the whole adjacency matrix in each iteration.

Sampling graphs from node embeddings. Once we have trained the embedding matrix $\boldsymbol{Z}$ we can sample graphs from $p(\boldsymbol{A}|\boldsymbol{Z})$ using 4.1. We adjust the sampled adjacency matrix by making it symmetric and setting the diagonal to zero, as we want the graphs to be undirected and to have no self-loops. We additionally remove the nodes with zero edges.

Evaluation strategy. In order to evaluate the ability of our embedding model to capture specific graph properties, we train embeddings on a single original graph and then we sample 100 graphs from the embeddings (from now on referred to as reconstructed graphs). For properties like number of triangles and clustering coefficient that have a single value per graph, we visualize the distribution of the property for the reconstructed graphs and compare it to the property of the original graph used for training. However, we will not able to exactly replicate the original property, even if the model is able to perfectly learn the expected adjacency matrix, because we have randomness in our procedure for reconstructing graphs from the embeddings. Having this in mind, in the experiments with synthetic graphs, we generate 100 graphs (from now on referred to as original graphs) from the graph generative model as well. We then compare the distribution of the property of interest for the original and reconstructed graphs. For properties like degree distribution and shortest path lengths, that represent a whole sequence for one graph rather than a single value, we compare only the original training graph and one reconstructed graph.

To better quantify the error of recovering the properties, we sometimes additionally used a version of MAPE (mean absolute percentage error) in which we take the absolute difference of the average property for the original and reconstructed graphs, divided by the average property of the original graphs (in case of experiments with synthetic graphs). In the case of experiments

with real graphs where we have only one original graph, we often used the absolute percentage error of the original property and the mean (or min/max) of properties of the reconstructed graphs. In both versions, we deal with a relative error measure which we can use to compare the results for both small and large values of the original property, which is not possible with absolute measures (for instance, an absolute error of 0.05 has a totally different meaning for a small clustering coefficient like 0.0001 and a large coefficient like 0.5).

4.3 Experiments on synthetic graphs

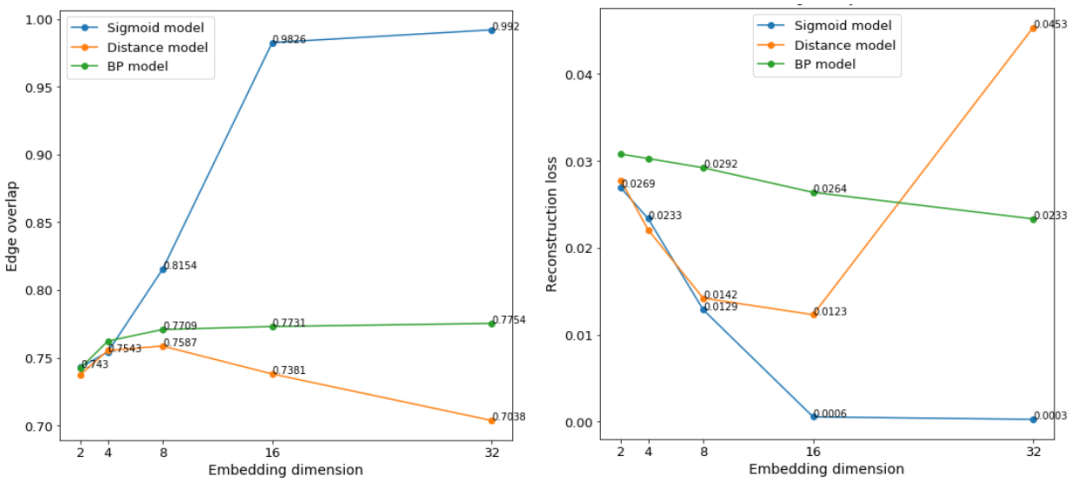
The use of synthetic graphs allows us to examine the graph properties separately and to better assess our ability of capturing them, as each of the used graph models generates simple graphs with specific characteristics.

In order to decide which embedding model to use, we performed the following experiment. We trained embeddings with dimensions $D \in \{2, 4, 8, 16, 32\}$ on ER graph ($N = 1000$, $p = 0.05$), using the sigmoid, Bernouli-Poisson and distance model (explained in Section 3.3). For each setting, we sample 100 graphs from the embeddings. As metrics for choosing the best embedding model we use the reconstruction loss (the final loss from training the embeddings) and the average edge overlap between the original and the reconstructed graphs. We calculate the edge overlap as a Jaccard similarity between the original and reconstructed adjacency matrix, and we take the average for all samples. We can see from Figure 4.1, that the sigmoid model has the best performance, as it has the highest edge overlap and lowest reconstruction loss in comparison with the other two methods. With 32 dimensions it achieves an edge overlap of 99%, which is much higher than the Bernouli-Poisson and the distance model. We performed the same experiment with other synthetic graphs and got similar conclusion. Therefore we decided to use the sigmoid model for the next experiments on synthetic graphs.

4.3.1 Sparsity

What is the effect of graph sparsity on the embeddings? In this section we are interested to find out if we can learn meaningful embeddings from sparse graphs and how the increase of sparsity affects the "quality" of the obtained embeddings.

To answer this question, we trained 2-dimensional embeddings using the sigmoid model for different types of synthetic graphs and levels of density. Figure



(a) Edge overlap

(b) Reconstruction loss

Figure 4.1: Comparison of the reconstruction loss and edge overlap for sigmoid, Bernoulli-Poisson and distance model trained on ER graphs ($N = 1000, p = 0.05$).

4.2 shows the embeddings for graphs with $N = 1000$ nodes, generated with ER, ER-DC, BA and WS models with low (0.002 - 0.003), medium (0.005 - 0.01) and high density (0.03 - 0.05). For a better illustration of the node degrees, the high degree nodes are colored in red, the low degree nodes are colored in green, while the rest are colored in blue. The embeddings for the dense graphs (the last row) have different shape for different types of graphs. For instance, the embeddings for the ER graph are points randomly scattered near the zero, while the embeddings for the scale-free graphs (ER-DC, BA) have triangular shape that clearly reflects the non-homogeneous degree distribution. However, the results for sparse graphs (the first row) were quite unexpected - instead of seeing the similar shapes with less points, the points are placed on a thin circle with empty center. This behavior is present for all graph types and their embeddings all look similar, which is not a desired outcome.

Why do we get this circular shape? The embeddings of this shape are actually in accordance with the definition of the sigmoid model, i.e. nodes are connected if the angle between them is small. As the graphs are sparse and the nodes do not have many connections, it was possible for the nodes to be ordered on a circle such that their neighbors are close to them and still lie on a circle. The optimized bias of the sigmoid model is negative for all the sparse graphs, meaning that two nodes will have high probability of connecting if

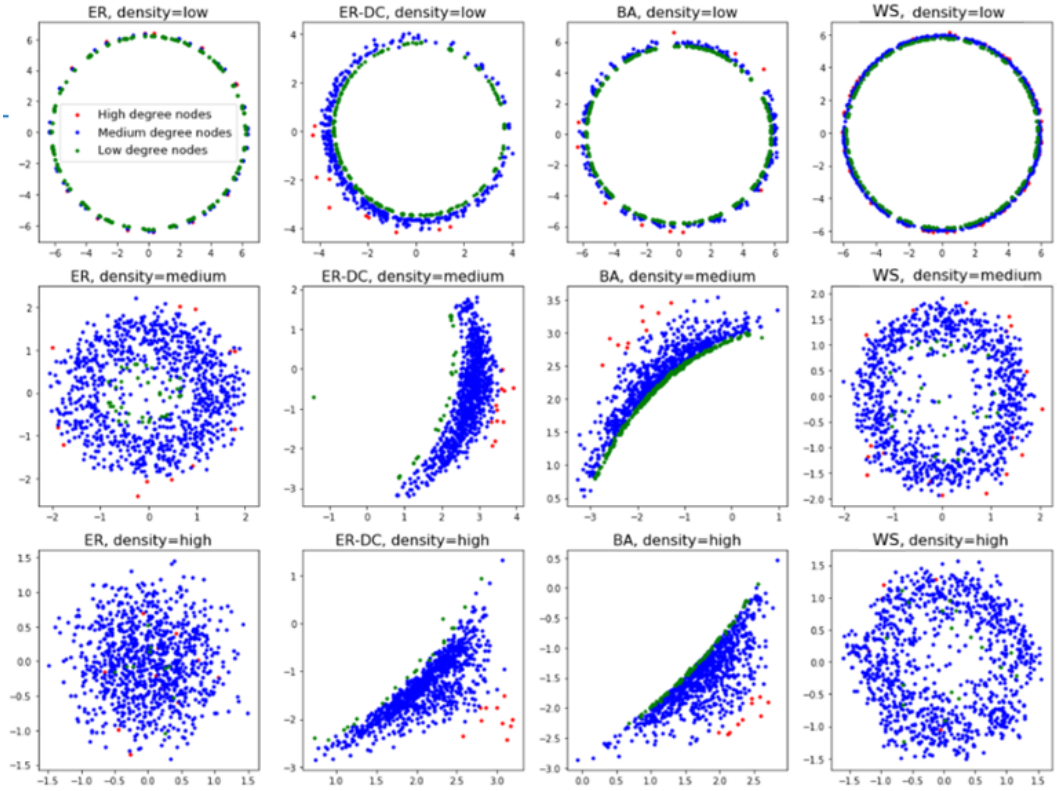


Figure 4.2: 2-dimensional embeddings for different types of graphs and levels of density

their embeddings are away from zero, hence the empty circle shape without points near zero. These observations justify the circular shape, however, it is obvious that the model overfits on sparse graphs and the embeddings do not learn true model from which the graphs are generated.

How can we solve the issue with sparsity? A way to deal with overfitting is to use regularization. Figure 4.3 shows the 2-dimensional embeddings for a sparse random graph ($N = 1000, p = 0.0005$), with different values for weight decay (L2 penalty). The increased regularization collapses the embeddings to (almost) zero, so effectively the only parameter that is learned is the bias. This makes sense as the ER model has only one parameter - the density p (for fixed N). Therefore, $\sigma(bias)$ should correspond to the effective density of the original graph (we cannot model the real density of the ER model, as we do not observe the nodes without edges in the training graph generated by it).

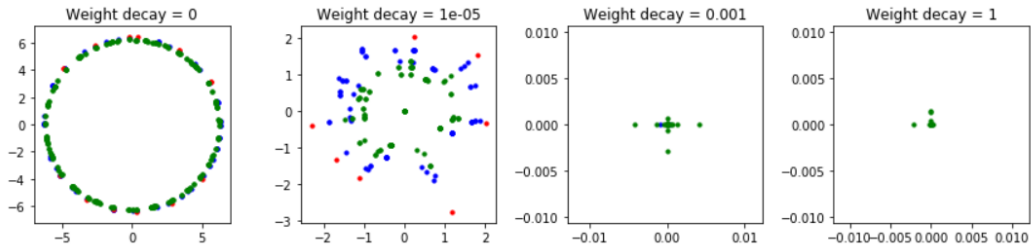


Figure 4.3: 2-dimensional embeddings trained on a sparse ER graph ($N = 1000, p = 0.0005$) using sigmoid model, with different regularization strengths.

We conclude that the sigmoid model learns dense graphs very well, but overfits for sparse models. Using regularization fixes this issue.

4.3.2 Degree distribution

Most of the real-world graphs are scale-free, meaning that their degrees follow a power law distribution, unlike random graphs whose degrees follow a Poisson distribution. Scale-free graphs usually contain a few hubs - nodes with large number of edges, while most of the nodes have low number of connections. In this part we are interested in answering the following question: **Can we reconstruct graphs with power law degree distribution?** To examine this, we generated graphs with ER-DC and BA models and trained node embeddings on them using the sigmoid model.

Figure 4.4 answers our question by comparing the degree distribution of the original graph generated by the ER-DC model and the one of the reconstructed graphs (sampled from the sigmoid model), for different values of α and c. Note that here we use only one original and one reconstructed graph per configuration. All plots are given on log-log scale for better visibility. The reason that the degrees of the original graph do not form a straight line (as we would expect for power law distributions) is the fact that we do clipping when sampling the adjacency matrix from the sigmoid model, i.e. if the sampled adjacency matrix contains entries bigger than 1, we clip them to exactly 1. The plots shows that even with 2-dimensional embeddings, the sigmoid model can capture the degree distribution of ER-DC graphs quite well.

Similarly, Figure 4.5 shows the degree distribution of the original and reconstructed BA graphs, for different values of m. The sigmoid model is capable of recovering the original degree distribution to some extent (the distributions

have the same slope in log-log scale). However, the model fails in recovering the minimum degree (which often corresponds to the parameter m) and in most cases where the minimum degree is larger than 1, it produces graphs with lower minimum degree than the original. However, the existence of minimum degree is an artifact of the BA model and it is unclear whether this problem would occur in real-world graphs. If we look at the cases where $m = 1$, we can conclude that the sigmoid model can capture the degree distribution reasonably well even in just 2 dimensions, while with 8 dimensions it can capture it almost perfectly.

How is the embedding shape reflecting the degree distribution? The 2-dimensional embeddings for dense ER-DC and BA graphs shown in the last row of Figure 4.2 have both triangular shape, unlike the other graphs whose embeddings have round shape. The high degree nodes (colored in red) are on the vertex of the triangle which is furthest away from zero, while the low degree nodes (colored in green) are on the opposite edge of the triangle. The justification for the triangular shape comes from exactly from the power law degree distribution - the high degree nodes are very few and they can be close together as they are probably connected. As the degrees get smaller and the nodes are less connected, the embeddings start to occupy more space. The nodes with the lowest degrees are rarely connected between each other which forces their embeddings to spread out, while at the same time they should not be too far from the high degree nodes, which leads to forming the triangular shape with the lowest-degree nodes on the opposite edge. The fact that high-degree nodes are connected to many of the other nodes, explains their high norm, as this makes the dot product large and the edge probability with other nodes very high.

As discussed in the previous subsection, the sigmoid model overfits on sparse graphs. In the case of ER-DC and BA graphs, the embeddings go from triangular to circular shape as we decrease the density. Figure 4.6 illustrates how regularization helps in the case of an ER-DC graph with $\alpha = 2$ and $c = 1$, for 2-dimensional embeddings. As we add regularization (weight decay = 1e-5), the embeddings start forming a line and, interestingly, the colors (green - blue - red) indicate that they are ordered by the degree. The norm of the the embeddings increases with the degrees of the nodes. The justification for this is the fact that in the "ground-truth" ER-DC model (and implicitly in the BA model) every node can be described perfectly using only its latent degree, which can be seen as one-dimensional embeddings.

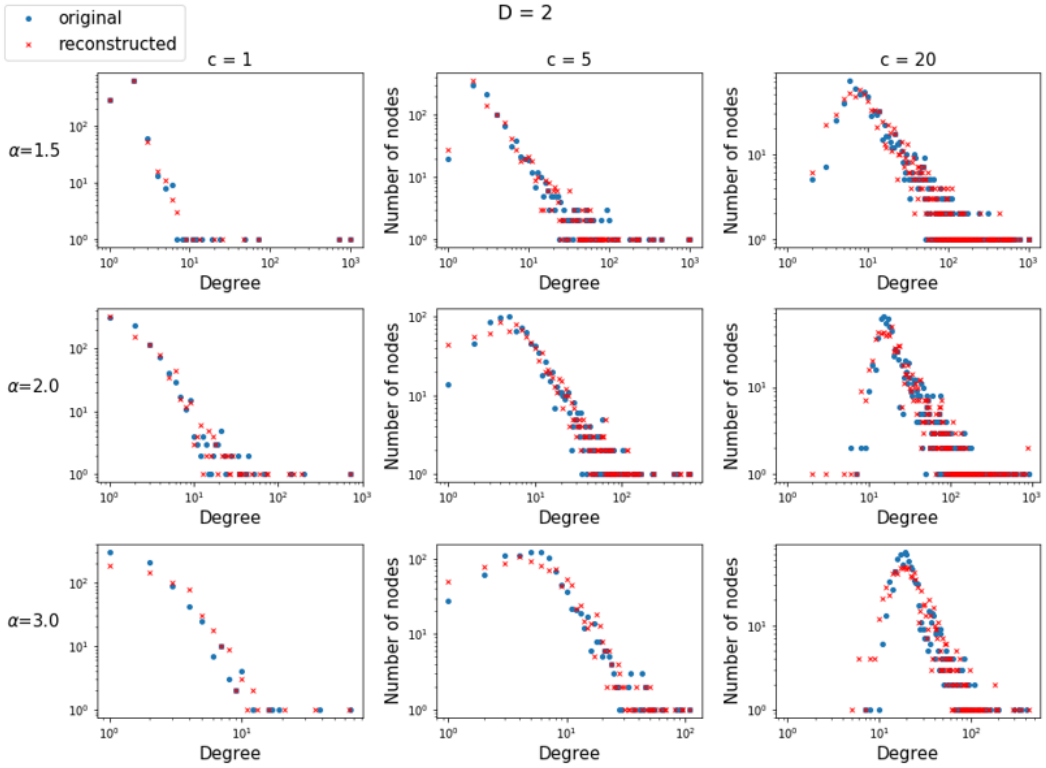


Figure 4.4: Degree distribution of original and reconstructed ER-DC graphs, for 2-dimensional embeddings and different values of α and c.

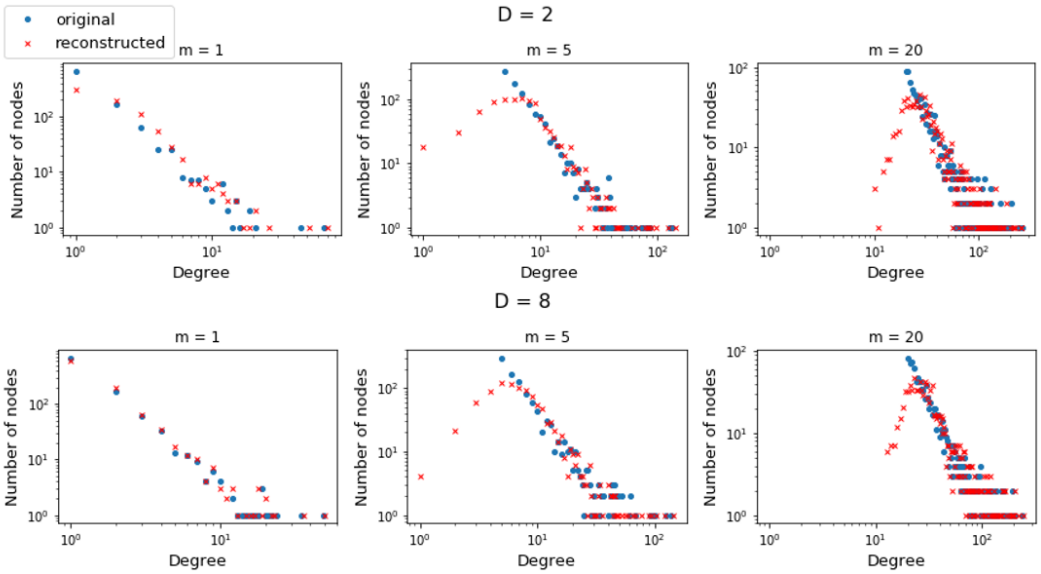


Figure 4.5: Degree distribution of original and reconstructed BA graphs, for different values of m and embedding dimensions 2 and 8.

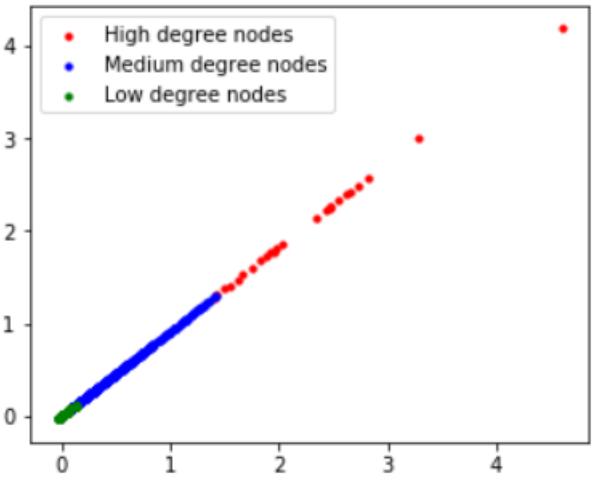


Figure 4.6: Effect of using regularization (weight decay = 1e-5) on 2-dimensional embeddings for ER-DC graph ($\alpha = 2$, $c = 1$)

4.3.3 Small world properties

Small-world graphs have interesting characteristics - the shortest paths between the nodes have small length and yet, the clustering coefficient is high, which is not the case with random graphs (where both shortest path lengths and clustering coefficient are small). **Can we learn graphs with small-world properties?** To answer this question we generated Watts-Strogatz graphs with different parameters and compared their properties with the properties of the graphs we reconstructed using the sigmoid model.

Figure 4.7 illustrates the distribution of the shortest path lengths for the original and reconstructed small-world graphs, for different values of p and K. Note that we use only one original and one reconstructed graph for each setting. As K is the number of nearest neighbours that the each node is initially connected to, increasing K leads to smaller shortest path lengths. The same happens when increasing p, as the increased randomness in the edges' placement decreases the distances between nodes. The figure shows that for embedding dimension 16, the distribution of shortest path lengths of the reconstructed graphs is quite close to the one from the original graph, meaning that the sigmoid model is capable of learning this property quite well.

We did the same experiments for the average clustering coefficient. As we increase p from 0 to 1, we move from a regular lattice, which has high clustering coefficient, towards a random graph, that has low clustering coefficient. Figure 4.8 compares the average clustering coefficient distribution of 100 original and 100 reconstructed graphs, for embeddings dimension 16 and different values of p and K. The clustering coefficient of the original graph used for training is shown as a red line. The third column shows the average clustering coefficients for WS graphs where 90% of the edges are randomly rewired ($p = 0.9$), making these graphs almost random. As expected, the clustering coefficient for these graphs is quite low (between 0 and 0.05). On the other hand the graphs with rewiring probability $p = 0.1$ have much higher clustering coefficients (between 0.35 and 0.55). This fact confirms the existence of the small-world effect for the WS graphs with small p: the clustering coefficient is much larger than for (almost) random graphs, yet they have low shortest path lengths, as it was shown from the previous experiment.

Figure 4.8 illustrates that the sigmoid model produces graphs with higher average clustering coefficient for graphs that are almost random ($p = 0.9$). In this case the absolute difference between the recovered and original clustering coefficients is between 0.01 and 0.03, which is quite large considering the range

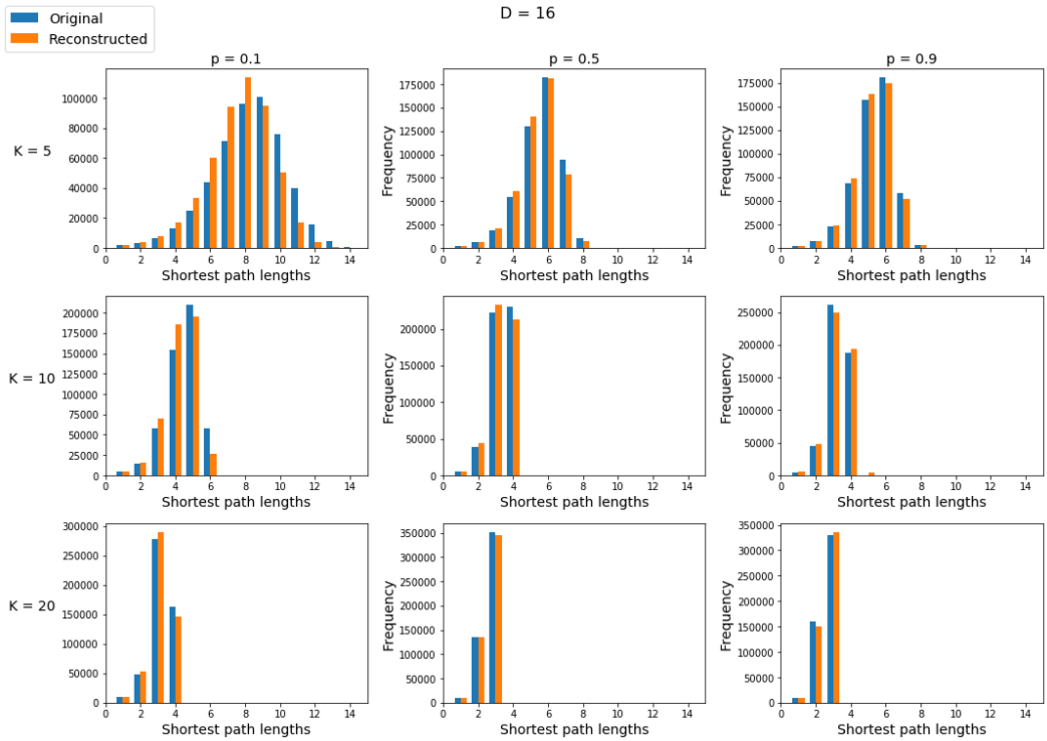


Figure 4.7: Shortest path lengths in original and reconstructed WS graphs, for different values of p and K. Embeddings dimension is 16.

of the coefficients of the original graphs. The absolute percentage error in this case is 344% in the worst case. However, we are more interested in small-world graphs (which have high clustering coefficient) rather than (almost) random graphs. For these graphs ($p = 0.1$) the sigmoid model is recovering the clustering coefficient much better. The absolute difference is 0.02 in the worst case, which is rather small error considering the original range of the coefficients. The absolute percentage error in the worst case is only 4.3%, which leads to the conclusion that the sigmoid model is able to capture the high clustering coefficient of small-world graphs reasonably well.

We can conclude from both experiments that we are able to capture small-world properties (low shortest path lengths and high clustering coefficient at the same time) using the sigmoid model.

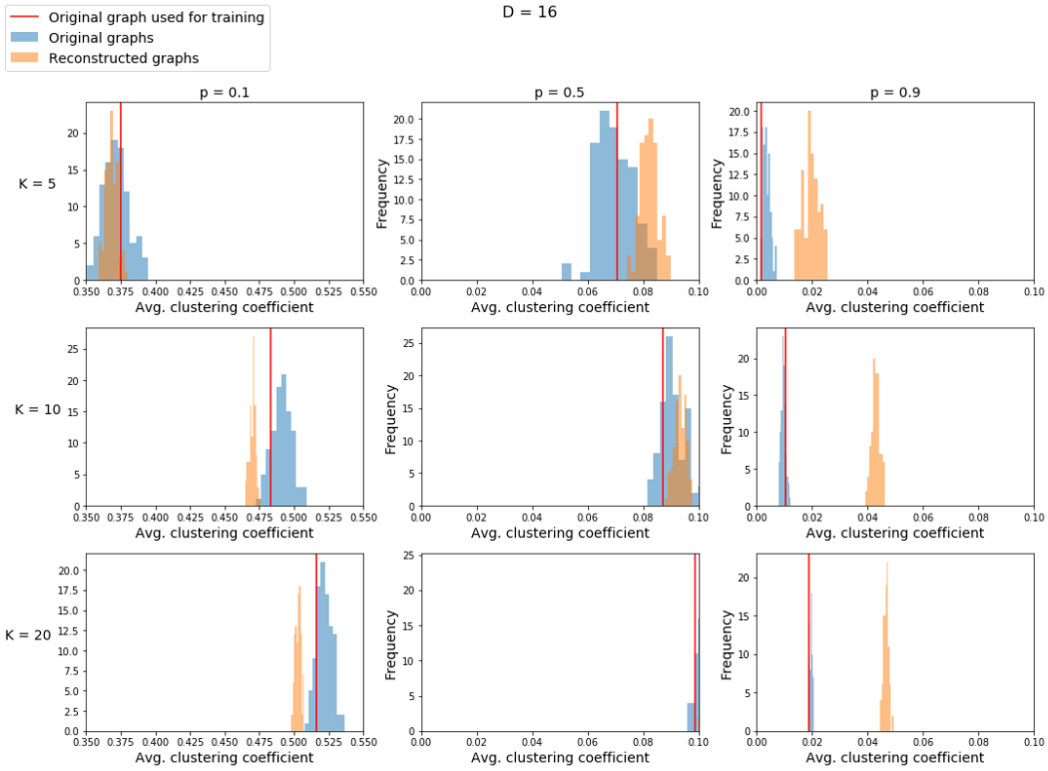


Figure 4.8: Average clustering coefficients in original and reconstructed WS graphs, for different values of p and K. Embeddings dimension is 16.

4.3.4 Communities

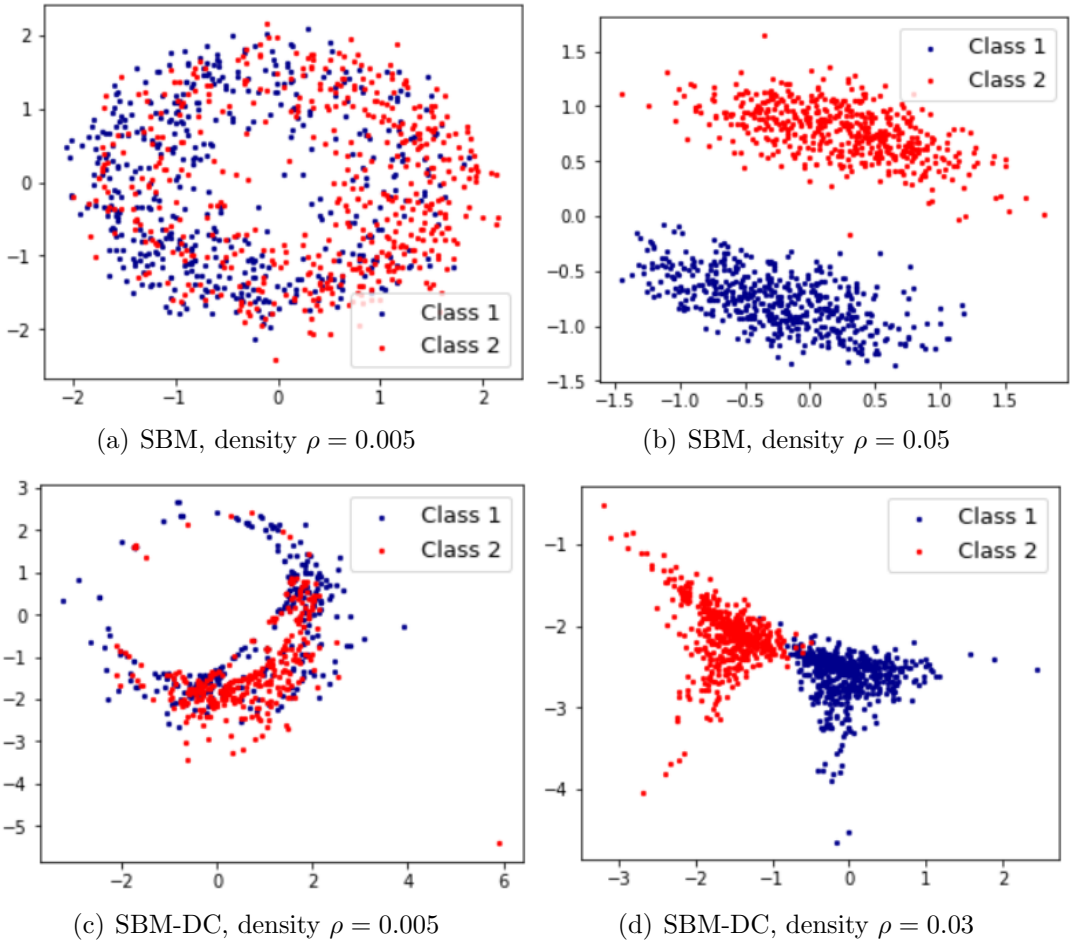
Can we capture the community structure of graphs? To answer this question, we generated graphs with 2 clusters using SBM and SBM-DC models. As an input to the SBM-DC model, we generated a degree sequence from power law distribution with $\alpha = 2.5$ and $c = 1$. We are interested to see whether 2-dimensional embeddings are enough to discover the 2 clusters and if they are well separated in the embedding space. Figure 4.9 shows the 2-dimensional embeddings of both dense and sparse SBM and SBM-DC graphs. The two original clusters/classes of the graphs (given with the class membership vector for the SBM and SBM-DC models) are shown in 2 colors: red and blue.

We can see from the figure that for the sparse SBM graph (within-cluster probability 0.007, between-cluster probability 0.002, effective density 0.005), the embeddings for the nodes of different clusters are not well separated, even though most of the blue points tend to be on the left, and the red on the right side. On the other hand, the embeddings for the dense SBM graph (within-cluster probability 0.07, between-cluster probability 0.02, effective density 0.05) are entirely separated, which is a great outcome. We observe similar results for the SBM-DC graphs: the sparse one has a circular shape and the clusters are not recognizable, while for the dense on we can clearly see two clusters shaped as triangles (due to the power law degree distribution). However, the inability to detect the clusters in the shown sparse graphs is not a limitation of the sigmoid model specifically, but rather due to a theoretical threshold to which 2 communities in SBM graphs can be detected by an algorithm, depending on the ratio of the within-cluster and between-cluster probabilities [16].

The previous experiments show that we can capture communities in SBM and SBM-DC graphs very well even with just two dimensions for the embeddings. However, for some values of the between-cluster and within cluster probabilities it is theoretically impossible to detect the clusters.

4.3.5 Number of triangles

In Random Geometric graphs two nodes are connected if their distance is smaller than a predefined distance threshold (radius). Since connectivity is determined by proximity, we expect an increase in the number of triangles in RGG as the radius gets larger. **Can we recover the number of triangles with the sigmoid model?** The following experiment was conducted in order to answer this question.



(a) SBM, density $\rho = 0.005$

(b) SBM, density $\rho = 0.05$

(c) SBM-DC, density $\rho = 0.005$

(d) SBM-DC, density $\rho = 0.03$

Figure 4.9: 2-dimensional embeddings for sparse and dense SBM and SBM-DC graphs. For dense graphs, the clusters are clearly separated even with 2 dimensions of the embeddings.

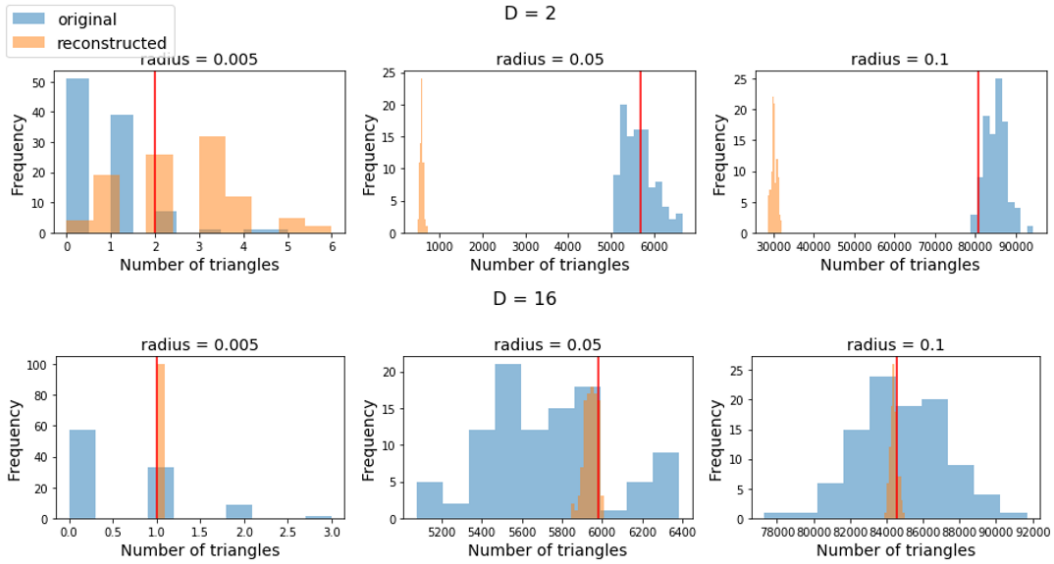
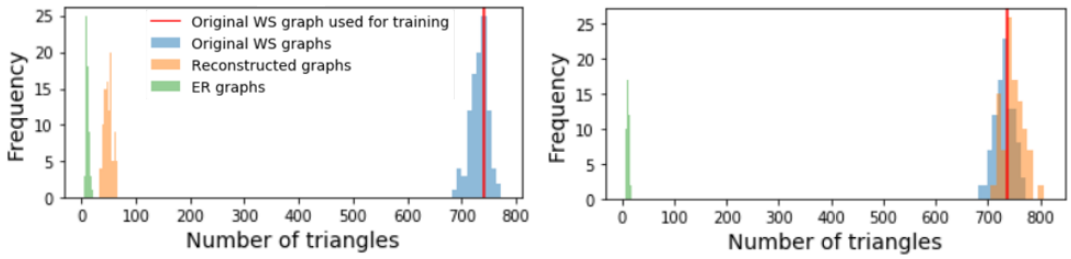


Figure 4.10: Number of triangles in original and reconstructed Random geometric graphs, for embedding dimensions 2 and 16.

We generated 100 original graphs from the RGG model for different values of the radius, chose one of those graphs to train the sigmoid model, and finally we used the trained sigmoid model to reconstruct 100 graphs. Figure 4.10 compares the distribution of number of triangles for the original and reconstructed graphs, for embedding dimensions 2 and 16 and varying radius. In each case we also show the number of triangles of the original graph used for training, as a single red line. It is worth noting that we always used 2 dimensional RGG model for generating the original graphs, regardless of the dimensionality of the embeddings trained on them, so that we have comparable results.

As expected, the number of triangles grows as we increase the radius. Using two-dimensional embeddings is not enough to capture the large number of triangles and the reconstructed graphs tend to have significantly lower number of triangles. However, with embeddings dimension 16, we are able to generate graphs whose range of number of triangles is within the range for the original ones. The range for the new graphs is much tighter and it is very close to the number of triangles of the original graph used for training.

Many real networks are sparse, with low diameter and high number of triangles at the same time (see Section 3.2.4). We are interested to see if we can capture the high number of triangles for such graphs as well. To answer this



(a) D = 2 (b) D = 8

Figure 4.11: Number of triangles in original and reconstructed sparse WS graphs (density $\rho = 0.004$) for embedding dimensions 2 and 8. The number of triangles in ER graphs with the same density is also shown for comparison.

question we generated sparse graphs (with density $\rho = 0.004$) using the WS model which is "specialized" in generating graphs with small-world properties. Figure 4.11 compares the number of triangles of the WS graphs, the reconstructed graphs from the embeddings, as well as for ER graphs with the same density. As expected, the WS graphs have much higher number of triangles than the ER graphs with the same density. The reconstructed graphs from 2-dimensional embeddings have also much smaller number of triangles than the WS graphs, however, with higher number of dimensions, the sigmoid model is capable of capturing the high number of triangles quite well.

We can conclude that we require higher number of dimensions for the embeddings in order to capture the number of triangles.

4.4 Experiments on real graphs

In this part we are interested to see how well we can capture the properties of real graphs using node embeddings. **As a first step, we would like to find out if we can reconstruct the original graph and how many dimensions for the embeddings are needed?**

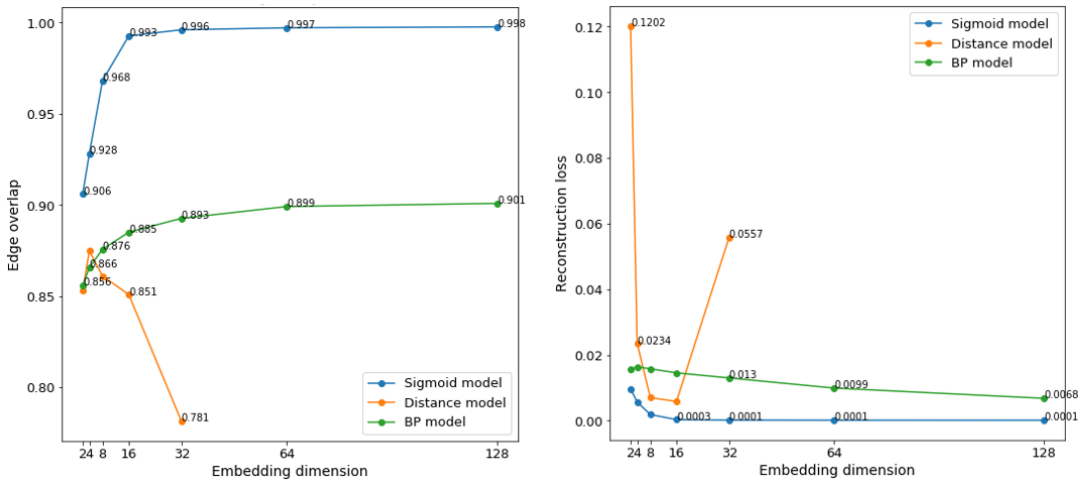
To answer this question, we trained embeddings on Cora-ML and Political Blogs datasets using the sigmoid, Bernoulli-Poisson and the distance model (explained in Section 3.3), for different embedding dimensions. For each setting, we sample 100 graphs from the embeddings. As evaluation metrics for the graph reconstruction we use the reconstruction loss (the final loss from training the embeddings) and the average edge overlap of the original and the

reconstructed graphs. We calculate the edge overlap as a Jaccard similarity between the original and reconstructed adjacency matrix, and we take the average for all samples. **With this experiment we also want to choose the embedding model with best performance, i.e. the model that achieves highest edge overlap and lowest reconstruction loss in least dimensions.**

Figure 4.12 compares the reconstruction loss and edge overlap for the three models trained on Cora-ML dataset, for up to 128 embedding dimensions. Same as in the case of synthetic graphs, the sigmoid model had the best performance and it achieved 99,6% edge overlap in 32 dimensions, which is much higher than the other two models. In fact, the Bernoulli-Poisson and the distance model showed 89,3% and 78,1% edge overlap, respectively. The distance model had convergence problems when trained on more than 32 dimensions, therefore we only show results for up to 32 dimensions for this model. We fixed this issue by adding regularization, however, the edge overlap was worse than the other two models. The reconstruction loss for the sigmoid model is decreasing until reaching 32 dimensions, after which it stays the same. Similarly, the increase in edge overlap for the sigmoid model is very small after 32 dimensions.

We have performed the same experiment for the Political Blogs dataset. Same as for the Cora-ML dataset, Figure 4.13 shows the edge overlap and reconstruction loss of the three models for the Political Blogs dataset. The sigmoid model shows best performance in this case as well, with 98.3% edge overlap with just two dimensions and 99.8% edge overlap in 32 dimensions. From this discussion we conclude that sigmoid model is the best choice and with already 32 dimensions it can recover both Cora-ML and Political Blogs graphs almost perfectly (edge overlap over 99.6%). Therefore, for the following experiments we will always use the sigmoid model.

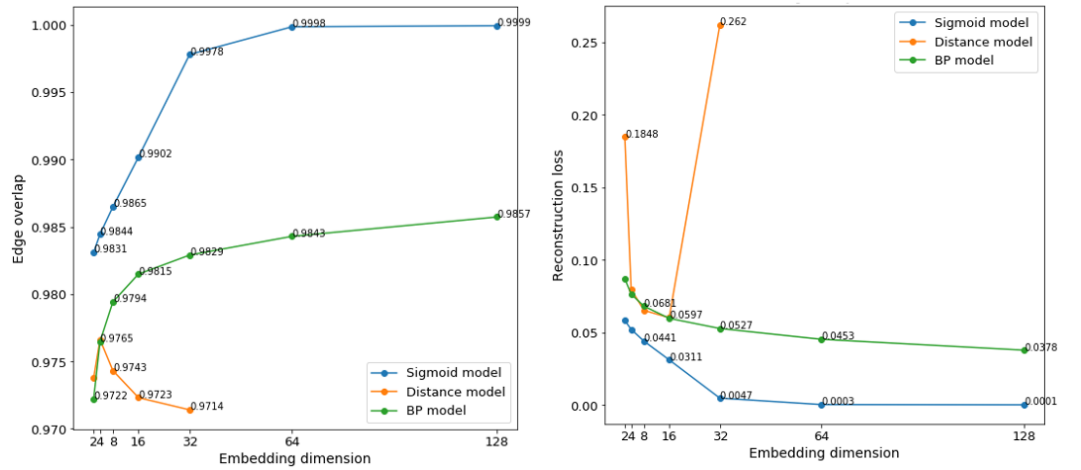
We are now interested to see how the embeddings look like for the two real graphs. Figure 4.14 shows 2-dimensional embeddings, trained on Cora-ML and Political Blogs using the sigmoid model. The clusters are illustrated in different colors. The embeddings for Cora-ML have a circular shape, with no points in the middle. As discussed in Section 4.3.1, this embedding shape is common for sparse graphs and it happens because of overfitting. Cora-ML is indeed a sparse graph, with density of 0.005, so this outcome is not surprising. For Political Blogs, the circular shape is less apparent, as this graph is denser than Cora-ML, and the embeddings look similar as the one for SBM-DC, reflecting the community structure and the degree heterogeneity.



(a) Edge overlap

(b) Reconstruction loss

Figure 4.12: Reconstruction loss and edge overlap for different embedding dimensions for Cora-ML.



(a) Edge overlap

(b) Reconstruction loss

Figure 4.13: Reconstruction loss and edge overlap for different embedding dimensions for Political Blogs.

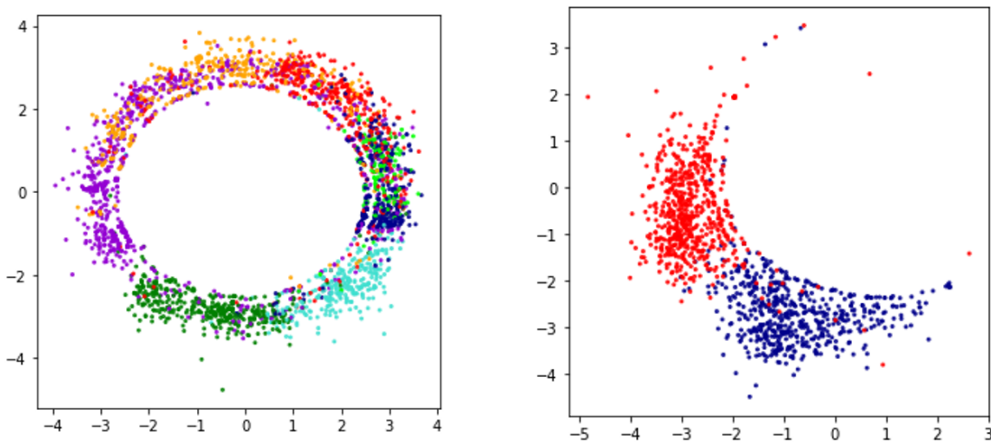
For both graphs, we can also see that the embeddings of the high-degree nodes have higher norm, same as in the case of synthetic graphs. These observations show that the embeddings of real graphs express the same behaviour as the embeddings of synthetic graphs.

Do the embeddings reflect the community stucture? Due to the sparsity of Cora-ML and the higher number of clusters, the clusters are not well separated in two dimensions, but we can still see that the points in the circle are ordered by the clusters to some extent. For Political Blogs we got better results and the embeddings shape shows the two clusters more clearly (even though they are not entirely separated).

For the other properties, we trained 32-dimensional embeddings for both datasets. Figure 4.15 compares the original properties (degree distribution, average clustering coefficient, shortest path lengths and number of triangles) of Cora-ML and Political Blogs dataset with the properties of the graphs reconstructed from the embeddings. We can conclude from the figure that for both datasets, the **degree distribution is almost perfectly recovered**. Regarding the number of triangles, for both datasets the reconstructed graphs tend to have larger number of triangles than the original graph. However, the MAPE is not very high (4.8% for Cora-ML and 6.8% for Political Blogs), so we can say that **the number of triangles is preserved reasonably well.**

Can we capture small world properties? We can see from Figure 4.15 that the average clustering coefficient of the reconstructed graphs is on average very close to the original one for both graphs, with 0.2% MAPE for Cora-ML and 4.7% MAPE for Political Blogs. We also get great reconstruction for the shortest path lengths for Political Blogs, while the reconstructed graph for Cora-ML has slightly smaller shortest paths than the Cora-ML graph. The figures also show that the small world properties are more apparent in Political Blogs, as it has higher clustering coefficient and lower shortest path lengths than Cora-ML.

The previous experiments show that we are able to learn the properties of real graphs quite well with the sigmoid model, but due to the complexity of real graphs we need higher dimensions for the embeddings in comparison to synthetic graphs.



(a) Cora-ML (b) Political blogs

Figure 4.14: 2-dimensional embeddings of Cora-ML and Political Blogs graphs, trained with sigmoid model.

4.5 Summary

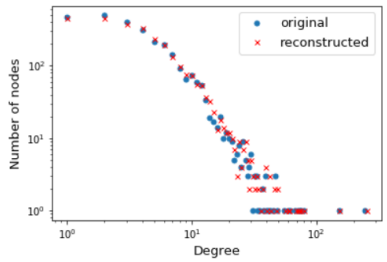
The goal of this chapter was to answer the following question: which properties can or cannot be captured with node embeddings? Our work led to the following surprising insights:

(i) Sparsity is a serious issue. The embeddings for sparse graphs all have a circular shape regardless of the characteristics of the graph, due to overfitting.

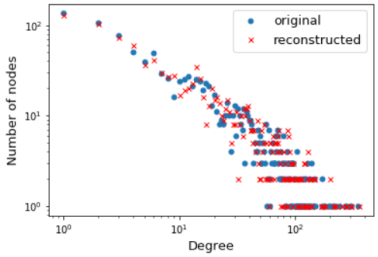
(ii) Embeddings are highly sensitive to regularization. Their shape changes drastically by changing the regularization strength.

(iii) Node embeddings do not capture all graph properties equally well. For instance, to capture the degree distribution it is enough to learn the latent degrees and we can do this reasonably well in just two dimensions.

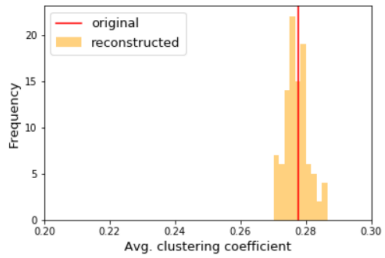
(iv) Node embeddings are able to capture the number of triangles even in cases of sparse graphs with large number of triangles. However, we need more dimensions to capture the triangles well.



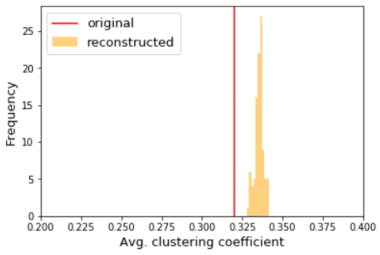
(a) Cora-ML: degree distribution



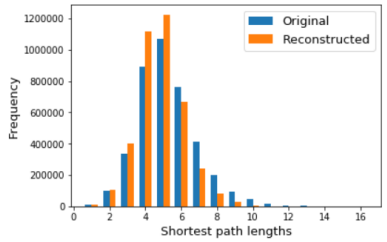
(b) Political blogs: degree distribution



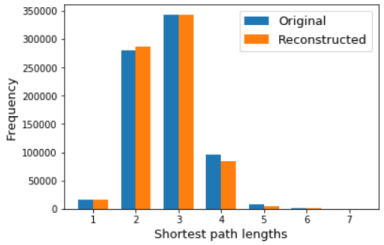
(c) Cora-ML: avg. clustering coef.



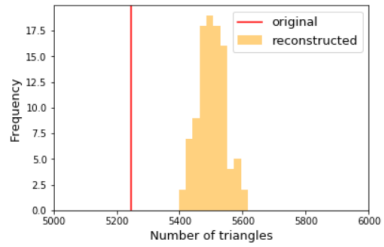
(d) Political blogs: avg. clustering coef.



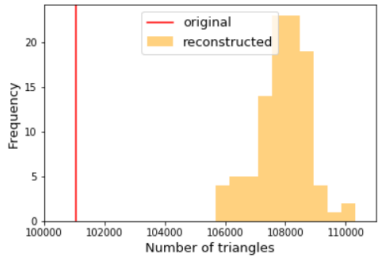
(e) Cora-ML: shortest path lengths



(f) Political blogs: shortest path lengths



(g) Cora-ML: number of triangles



(h) Political blogs: number of triangles

Figure 4.15: Properties of Cora-ML and Political Blogs compared to the properties of the graphs reconstructed from their 32-dimensional embeddings.

Chapter 5

Framework for graph generation

5.1 Idea

In the previous chapter, we have empirically shown which properties can or cannot be captured with node embeddings. Now that we have a better understanding of the capabilities and limitations of simple embedding approaches, our main goal is to answer the following question: **Can we use node embeddings for graph generation?**

Our idea for a graph generation framework based on node embeddings is illustrated in Figure 5.1. Consider an original graph G with adjacency matrix $\boldsymbol{A}$. The first step is to train an embedding matrix $\boldsymbol{Z}$ from G by maximizing $p(\boldsymbol{A}|\boldsymbol{Z})$. Second, we perform density estimation on the trained embeddings. To generate new graphs, we follow the opposite direction: we sample new embeddings $\tilde{\boldsymbol{Z}}$ from the estimated density $p(\boldsymbol{Z})$ and then we use the embeddings to sample a new graph $\tilde{G} \sim p(\boldsymbol{A}|\tilde{\boldsymbol{Z}})$.

The motivation behind the approach of performing density estimation on the embeddings, comes from the observation that the embeddings do not span the whole latent space, but they rather have a specific shape that reflects the graph properties. For instance, we have shown in Section 4.3 that a triangular shape indicates a graph with power-law degree distribution. If the embeddings are randomly sampled from the whole latent space, the reconstructed graphs probably would not have realistic properties. By doing density estimation on the embeddings, we want to estimate the true generative process of the original graph and by sampling from the estimated density we expect to generate novel graphs with similar properties to the original.

5.2 Experimental setup

We performed our experiments on Cora-ML and Political Blogs datasets with the same preprocessing as in Section 4.2. We trained the embeddings using the sigmoid model and used the same training and sampling procedures as in Section 4.2.

Density estimation. We used two methods for density estimation, GMM and B-NAF (explained in more detail in Section 3.4). For both methods, 90% of the embeddings are used for training ($\boldsymbol{Z_{train}}$) and the other 10% for validation ($\boldsymbol{Z_{val}}$). In the case of GMM, we set the number of components to the number of clusters of the original graph (both Cora-ML and Political Blogs are labeled). For density estimation with B-NAF we had the following setup: we used Gaussian distribution as a base density and we applied a single flow of B-NAF, consisting of two layers with 8 hidden nodes each. We experimented with higher number of layers and nodes, as well as with stacking multiple B-NAF flows. However, this was not necessary as a single B-NAF flow with two layers was already flexible enough to model complex densities. Using a more complex architecture often led to overfitting and by increasing the regularization we were getting similar results as with the simple flow architecture.

To be able to evaluate the density as well as to sample from it, we need both the flow f_k and its inverse f_k^{-1}. Since B-NAF does not have a closed-form inverse and we need to perform density evaluation more often than sampling (in order to calculate the loss in each iteration while training), we parametrize the flow with inverse transformations f_k^{-1}. Then, for calculating $\log p(\boldsymbol{z_K})$, as we go backwards in the flow to the base distribution, we apply $\boldsymbol{z_{k-1}} = f_k(\boldsymbol{z_k})$ instead of the $\boldsymbol{z_{k-1}} = f_k^{-1}(\boldsymbol{z_k})$. To train the B-NAF flow we minimized the following training loss:

$$\mathcal{L}(\boldsymbol{Z_{train}}) = -\log p(\boldsymbol{Z_{train}}), \tag{5.1}$$

where $\log p(\boldsymbol{Z_{train}})$ was obtained as in Equation 3.17. We trained the parameters of the flow with Adam optimizer with learning rate of 0.05. We also used early stopping with patience of 100 iterations, meaning that the training stops if the validation loss (given with $-\log p(\boldsymbol{Z_{val}})$) does not decrease in 100 iterations. By looking at the training and validation loss over the training iterations, we observed that the model is suffering from serious overfitting in high dimensions. In order to solve this issue, we used different regularization strengths for low and high dimensions. For 2 and 8 dimensions we used weight decay of 1e-6 and 1e-4 respectively, while for larger dimensions (for instance

32) we applied weight decay of 0.01 and additionally added random noise of (-0.01, 0.01) to the training embeddings.

Sampling embeddings from the estimated density. As we have seen above, in order to evaluate the density we were using f_k in each step while going backwards in the flow. To sample from the transformed density, we need to go in the opposite direction. In fact, we only know how to sample from the base distribution $\boldsymbol{z}_0 \sim q_0(\boldsymbol{z}_0)$, however, we can use that sample and by successively applying the inverse transformation $\boldsymbol{z_k} = f_k^{-1}(\boldsymbol{z_{k-1}})$ we can transform it into a sample from the transformed distribution $\boldsymbol{z}_K \sim q_K(\boldsymbol{z}_K)$ (the only difference with the setting explained in Section 3.4.1 is that we construct the flow by chaining the inverse of f_k^{-1} instead of f_k).

We use numerical inversion to compute $\boldsymbol{x} = f^{-1}(\boldsymbol{y})$, by looking the problem as finding a root of the multivariate function $f(\boldsymbol{x}) - \boldsymbol{y}$, for a given $\boldsymbol{y}$. We use the Powell's hybrid method (also known as "Dog Leg" method) [28] which is a combination of Newton's method and the steepest descent method. By looking at the 2-dimensional embeddings sampled form the estimated density using this method, we noticed that we sometimes receive "bad" samples, i.e. samples that do not come from the estimated distribution, but they are rather the initial guesses used for the root finding, which happens when the hybrid method fails to converge. The reason why this problem occurs is the existence of flat regions with no density, as for instance the region inside the circle shown in Figure 5.4. In order to model such regions with no density, the transformation f needs to have very steep regions to "push the density apart". Because of the steepness of the function in these places, numerical inversion is very unstable and we need very high precision / low tolerance to invert the function exactly. Sometimes this fails and we obtain "bad" samples near these problematic steep regions. Moreover, we need higher dimensionality for the embeddings and in this case the sampling becomes even more challenging since numerical root finding is in general problematic in higher dimensions.

To fix this issue and to still be able to use B-NAF in our graph generation framework, we performed a correction of the sampling by rejecting the "bad" samples. If we get a sample $\tilde{\boldsymbol{z}}_{\boldsymbol{i}}$, such that $\log p(\tilde{\boldsymbol{z}}_{\boldsymbol{i}}) < t$, where t is a predefined threshold, then we reject the sample and we repeat this step until getting a good sample. Figure 5.6 shows the how the sampled embeddings look like with and without sampling correction in the case of 2-dimensional embeddings of Cora-ML. For lower dimensions this approach was working fine and we could use higher thresholds (e.g. threshold = -6 for D = 2), but as we increase the embeddings dimensionality, it gets more difficult and slower

to find samples that satisfy our condition and was even impossible to do it in reasonable time for high dimensions like 32. For this reason, we decided to use only 8-dimensional embeddings for the experiments, even though we are aware that for capturing certain properties we need embeddings of higher dimensionality (as discussed in Section 4.3). Additionally, we relaxed the threshold for correcting the sampling to t = -12 to have a more efficient sampling.

With this approach we were able to generate graphs from Political Blogs, however, the sampling correction was still extremely slow for Cora-ML (due to its size). Therefore, the experiments on Political Blogs were done with sampling correction, whereas the experiments on Cora-ML were performed without correction.

Evaluation strategy. To evaluate the quality of the generated graphs, we structured the experiments in the following way. For each original graph, we train an embedding matrix $\boldsymbol{Z}$ and we estimate the density $p(\boldsymbol{Z})$. Then, we sample 10 embedding matrices from $p(\boldsymbol{Z})$, and from each of them we sample 10 graphs. In this way we end up with 100 generated graphs. Now we want to see whether the properties of the generated graphs are similar to the ones of the original graph used for training. For properties having a single value per graph, like edge and triangle count, clustering coefficient and average shortest path lengths, we compare the original property and the range of that property in the generated graphs. We also compute a relative error as a version of MAPE, by taking the absolute difference of the original property and the average property of the generated graphs, divided by the original property. For properties that have a whole sequence per graph, such as the degree distribution, we plot the property of the original and only one generated graph (same as in Section 4.2).

5.3 Results

The first question that we want to address is **how well can we estimate the density for the embeddings**? To examine this visually, we trained 2-dimensional embeddings on Political Blogs and Cora-ML graphs and we perfomed density estimation on them using GMM and B-NAF. Figure 5.2 illustrates the 2-dimensional embeddings of Political Blogs, the estimated density with GMM and the sampled embeddings from that density. We can see that the estimated density does not match the shape of the embeddings at all. In fact, GMM approximates the density with two ellipses, while the original embeddings have a more complex shape, looking like two triangles forming a half-circle. On the other hand, B-NAF showed outstanding performance in

estimating the density. As shown in Figure 5.3, the shape of the estimated density is very similar to the shape of the original embeddings. Additionally, the high-density regions are indeed the places having more points in the original embeddings, and in this case they are indicating the existence of two clusters.

Figure 5.4 and Figure 5.3 show the estimated density for the 2-dimensional embeddings of Cora-ML with GMM and B-NAF respectively. We can draw the same conclusion - GMMs are not flexible enough to estimate the complex shapes of the embeddings, while B-NAF is a great choice for this task. In this case, B-NAF was able to capture the circular shape of Cora-ML embeddings as well as to respresent the high-density regions correctly.

The next question that we want to answer is **whether we can use node embeddings to generate graphs with realistic properties** (i.e. similar properties to the ones of the original graph)? For this purpose, we trained 8-dimensional embeddings for Political Blogs and Cora-ML and estimated their densities with both GMM and B-NAF. Table 5.1 summarizes the results for the Political Blogs dataset. The first row show the edge count, triangle count, average clustering coefficient and the average shortest path length of the original Political Blogs graph. The second row shows the range of these properties in the graphs generated by estimating the density with GMM, as well as the MAPE between the original and the generated graphs (in brackets). Similarly, the third row summarizes the properties of the graphs generated by using B-NAF for density estimation.

We can observe that GMM produces graphs with 64% more edges on average than the original graph, while the edge count in B-NAF graphs is much closer to the original (17% less edges on average). Moreover, GMM generates graphs with over 4 times more triangles on average, whereas B-NAF underestimates the number of triangles by 30%, which is a much better approximation. The average clustering coefficient and average shortest path lengths for B-NAF graphs are also much closer to the original ones, in comparison to the GMM graphs. From this discussion we can conclude that the properties of the graphs generated by using B-NAF show higher similarity with the properties of the original graph, than the ones generated by using GMM for the embeddings' density. This is expected since B-NAF is a much better choice for density estimation than GMM, as discussed above. Even though B-NAF achieved better results than GMM, its results could be better, especially for the number of triangles which were underestimated with 30% MAPE. However, these results should not be surprising as we only used 8 dimensions for the embeddings (due to the problem with numerical inversion), but we need more dimensions

to capture some properties such as the number of triangles (as shown in Section 4.3.5).

Similarly as for the Political Blogs dataset, Table 5.2 compares the properties of Cora-ML with the properties of the generated graphs with GMM and B-NAF. B-NAF again achieves better MAPE than GMM for all the properties, yet, both approaches have much worse results in comparison with the respective results for Political Blogs. In fact, the number of triangles is overestimated by 246% on average with B-NAF. However, there is a possible reason for the unsatisfactory results. Not only do we use smaller number of dimensions than needed (as discussed above), but we also do not correct the sampling in the case of Cora-ML. In this case, due to the problems with numerical inversion, we sometimes deal with bad samples, i.e. embeddings that do not come from the modeled density, which in turn produce graphs with quite different properties than the original.

5.4 Summary

The goal of this chapter was to see whether we can generate graphs from node embeddings. Our idea is to perform density estimation on the embeddings and to generate new graphs by sampling embeddings from that density. We have shown that GMM is a poor choice for estimating the density as it lacks flexibility, while B-NAF is powerful enough to estimate the density for embeddings of any complex shape.

Even though it is an excellent density estimator, B-NAF has a major drawback that seriously affects the quality of our generated graphs - the lack of closed-form inverse. In order to sample from the estimated density, we need the inverse of the flow, and because of this limitation of B-NAF we need to rely on numerical inversion which is unstable and inefficient in higher dimensions. To show that our approach with density estimation is still promising, we used correction for the sampling. The graphs generated with corrected sampling show reasonably good properties (considering the lower number of dimensions), whereas in the cases of no correction, the properties of the generated graphs are quite different than the ones of the original graph.

We can conclude that this approach for generating graphs from node embeddings looks promising, however, it faces considerable issues that need to be resolved in order to have a successful generative model. From one side, we often need more dimensions for the embeddings to capture all properties,

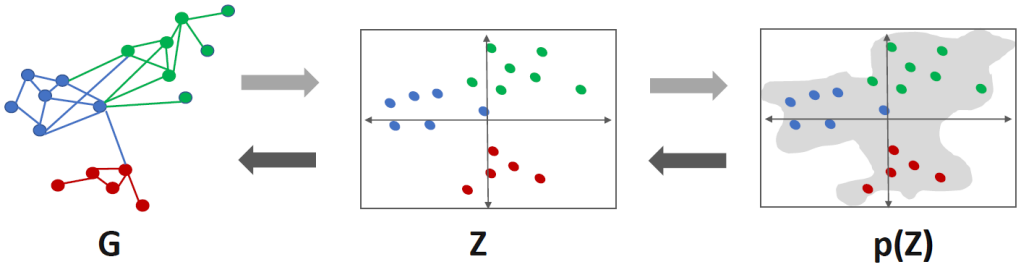


Figure 5.1: The main idea for our graph generation framework. We start with an original graph G, for which we train an embedding matrix $\mathbf{Z}$. Once we have the embedding matrix, we approximate the density $p(\boldsymbol{Z})$ of the embeddings by using a method for density estimation. To generate new graphs, we sample embeddings from $p(\boldsymbol{Z})$ and then we use the embedding model to reconstruct graphs from the embeddings.

while on the other side density estimation and sampling in higher dimensions turned out to be challenging. It would be interesting to see whether we can overcome these challenges as a future work.

Graph	Edge count	Triangle count	Avg. clustering coeff.	Avg. shortest path length
Original	16K	101K	0.32	2.74
GMM	24K - 31K (64%)	349K - 583K (335%)	0.5 - 0.58 (67%)	2.35 - 2.5 (11%)
B-NAF	12K - 15K (17%)	50K - 96K (30%)	0.2 - 0.28 (23%)	2.8 - 2.9 (4.5%)

Table 5.1: Comparison of the properties of Political Blogs and the graphs generated by sampling 8-dimensional embeddings from the estimated density with GMM and B-NAF. The first row shows the properties of the original graph. The second and third row show the range of the properties of the generated graphs, followed by the MAPE between the original and generated properties (in brackets), for GMM and B-NAF respectively.

Graph	Edge count	Triangle count	Avg. clustering coeff.	Avg. shortest path length
Original	8K	5K	0.27	5.27
GMM	26K - 27K (236%)	100K - 104K (1848%)	0.44 - 0.46 (64%)	3.08 - 3.11 (41%)
B-NAF	12K - 13K (60%)	17K - 19K (246%)	0.33 - 0.36 (25%)	3.38 - 3.44 (35%)

Table 5.2: Comparison of the properties of Cora-ML and the graphs generated by sampling 8-dimensional embeddings from the estimated density with GMM and B-NAF. The first row shows the properties of the original graph. The second and third row show the range of the properties of the generated graphs, followed by the MAPE between the original and generated properties (in brackets), for GMM and B-NAF respectively.

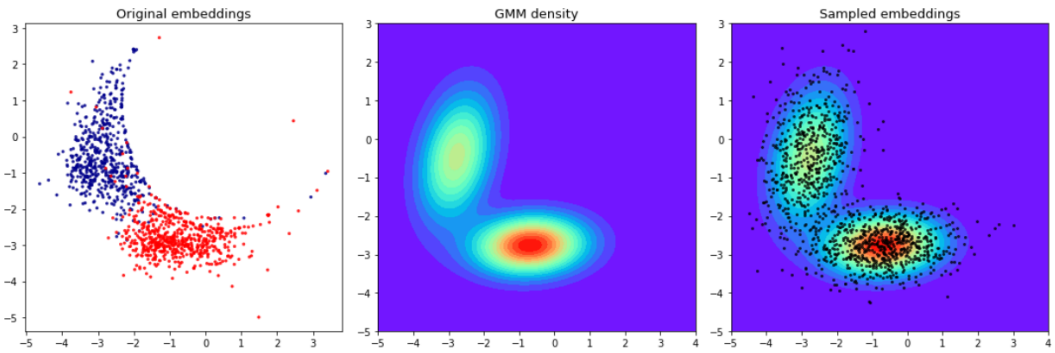


Figure 5.2: The left plot illustrates the 2-dimensional embeddings of Political Blogs, trained with the sigmoid model. The middle plot shows the density estimated with GMM, while the right plot additionally shows the new embeddings sampled from the estimated density.

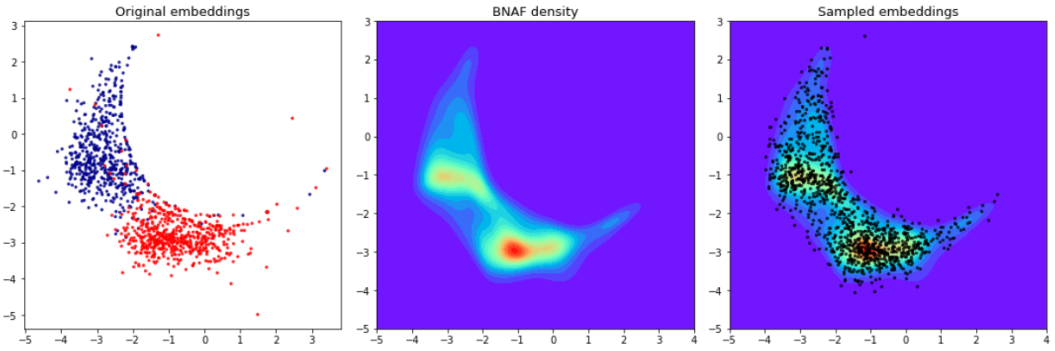


Figure 5.3: The left plot illustrates the 2-dimensional embeddings of Political Blogs, trained with the sigmoid model. The middle plot shows the density estimated with B-NAF, while the right plot additionally shows the new embeddings sampled from the estimated density.

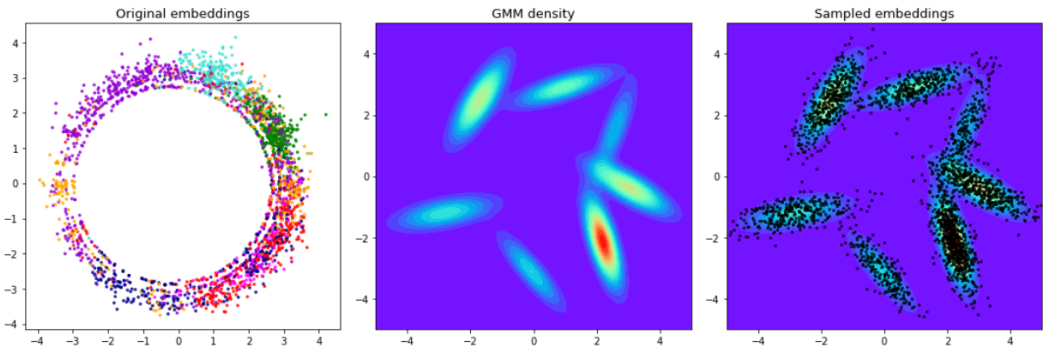


Figure 5.4: The left plot illustrates the 2-dimensional embeddings of Cora-ML, trained with the sigmoid model. The middle plot shows the density estimated with GMM, while the right plot additionally shows the new embeddings sampled from the estimated density.

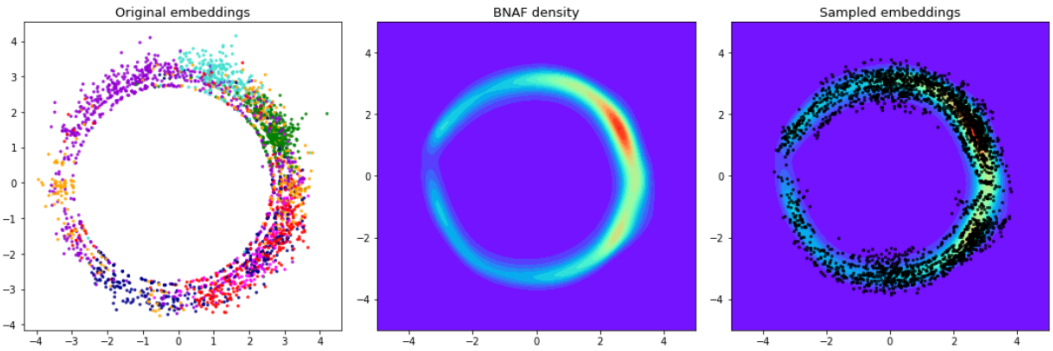
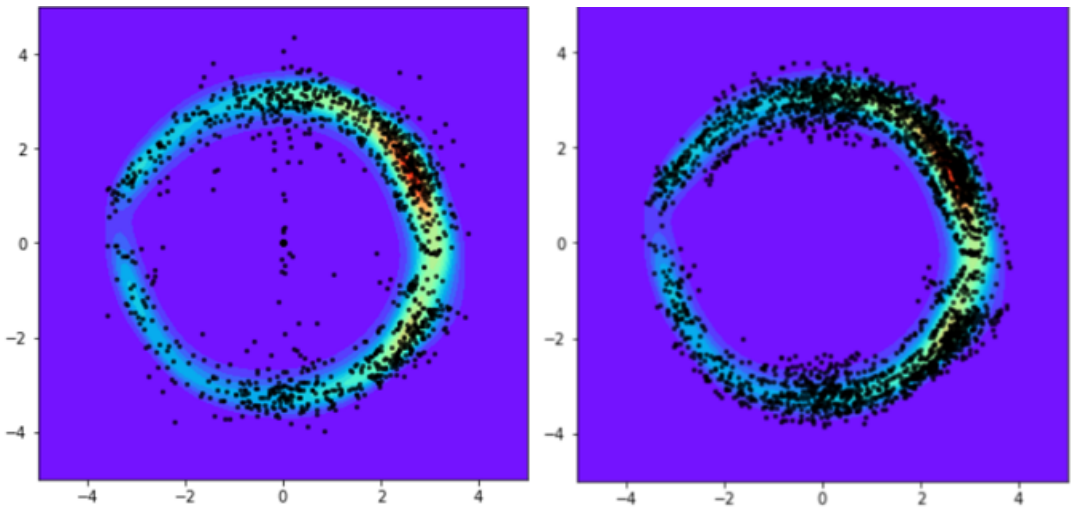


Figure 5.5: The left plot illustrates the 2-dimensional embeddings of Cora-ML dataset, trained with the sigmoid model. The middle plot shows the density estimated with B-NAF, while the right plot additionally shows the new embeddings sampled from the estimated density.



(a) Sampling without correction

(b) Sampling with correction

Figure 5.6: The plots show the sampled embeddings from the estimated density (using B-NAF) for the 2-dimensional embeddings of Cora-ML. The sampling in the left plot is done without correction, while in the right plot we correct the sampling by rejecting the "bad" samples.

Chapter 6

Conclusion and Future work

In this thesis we empirically examined the capabilities and limitations of node embeddings in (i) capturing graph properties and (ii) generating graphs.

Our work led to the following surprising insights. First, we found out that it is difficult to learn meaningful embeddings from sparse graphs. The embeddings tend to have circular shape regardless of the graph's characteristics, as a result of overfitting. Second, we have seen that the embeddings are highly sensitive on regularization. Their shape changes drastically by changing the regularization strength. Last, we conclude that node embeddings do not capture all graph properties equally well. Properties like the degree distribution can be captured reasonably well even in two dimensions, while for capturing the number of triangles we need embeddings in higher dimensions.

Furthermore, we showed that performing density estimation on the embeddings and sampling from it can be a promising approach for graph generation. However, density estimators like (Block) neural autoregressive flows might not be the right choice yet. Even though they are expressive enough to model any complex density, the lack of closed-form inverse and the instability of numerical inversion in higher dimensions, makes the sampling from the estimated density impractical.

There are several directions in which our work can be extended in the future:

(i) Theoretically prove the insights we discovered empirically, regarding the ability of node embeddings to capture graph properties and their limitations that come as a result of the i.i.d. assumption.

(ii) Solve the problem with sampling from the embeddings' density in higher dimensions. One way would be to try out another density estimation

method that is both flexible and has closed-form inverse.

(iii) Properly estimate the density of the embeddings with Variational inference. In our approach we first train the embeddings with maximum likelihood and then we perform density estimation on the trained embeddings. A proper way of learning the density would be to assume a prior for the embeddings and maximize the posterior density. As the later is intractable, we can use methods from Variational inference to approximate it. However, the success of this approach is questionable as the choice of prior has a huge effect on the embeddings (for instance, we have shown that the embeddings are highly sensitive to regularization).

(iv) Learn from multiple graphs. Our proposed approach for graph generation uses a single graph for training. It would be interesting to see how this approach extends to multiple training graphs, by introducing a graph-level latent feature.

Bibliography

[1] Lada A. Adamic and Natalie Glance. The political blogosphere and the 2004 u.s. election: Divided they blog. In *Proceedings of the 3rd International Workshop on Link Discovery*, LinkKDD '05, pages 36–43, New York, NY, USA, 2005. ACM.

[2] Albert-Laszlo Barabasi and Reka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.

[3] Albert-László Barabási and Márton Pósfai. *Network science*. Cambridge University Press, Cambridge, 2016.

[4] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic*, NIPS'01, pages 585–591, Cambridge, MA, USA, 2001. MIT Press.

[5] Smriti Bhagat, Graham Cormode, and S. Muthukrishnan. Node classification in social networks. *CoRR*, abs/1101.3291, 2011.

[6] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[7] Nicola De Cao, Ivan Titov, and Wilker Aziz. Block neural autoregressive flow. *35th Conference on Uncertainty in Artificial Intelligence (UAI19)*, 2019.

[8] P. Erdős and A Rényi. On the evolution of random graphs. In *PUBLICATION OF THE MATHEMATICAL INSTITUTE OF THE HUNGARIAN ACADEMY OF SCIENCES*, pages 17–61, 1960.

[9] Aditya Grover and Jure Leskovec. Node2vec: Scalable feature learning for networks. In *Proceedings of the 22Nd ACM SIGKDD International*

Conference on Knowledge Discovery and Data Mining, KDD '16, pages 855–864, New York, NY, USA, 2016. ACM.

[10] Aditya Grover, Aaron Zweig, and Stefano Ermon. Graphite: Iterative generative modeling of graphs. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, pages 2434–2444, 2019.

[11] William L. Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. *CoRR*, abs/1709.05584, 2017.

[12] Peter D. Hoff, Adrian E. Raftery, and Mark S. Handcock. Latent space approaches to social network analysis. *JOURNAL OF THE AMERICAN STATISTICAL ASSOCIATION*, 97:1090–1098, 2001.

[13] Paul Holland, Kathryn Laskey, and Samuel Leinhardt. Stochastic blockmodels: First steps. *Social Networks - SOC NETWORKS*, 5:109–137, 06 1983.

[14] Chin-Wei Huang, David Krueger, Alexandre Lacoste, and Aaron C. Courville. Neural autoregressive flows. *CoRR*, abs/1804.00779, 2018.

[15] Brian Karrer and M.E.J. Newman. Stochastic blockmodels and community structure in networks. *Physical review. E, Statistical, nonlinear, and soft matter physics*, 83:016107, 01 2011.

[16] Tatsuro Kawamoto, Masashi Tsubaki, and Tomoyuki Obuchi. Mean-field theory of graph neural networks in graph partitioning. *CoRR*, abs/1810.11908, 2018.

[17] Thomas Kipf and Max Welling. Variational graph auto-encoders. *ArXiv*, abs/1611.07308, 2016.

[18] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907, 2016.

[19] Thomas N. Kipf and Max Welling. Variational graph auto-encoders. *CoRR*, abs/1611.07308, 2016.

[20] Yujia Li, Oriol Vinyals, Chris Dyer, Razvan Pascanu, and Peter W. Battaglia. Learning deep generative models of graphs. *CoRR*, abs/1803.03324, 2018.

[21] Renjie Liao, Yujia Li, Yang Song, Shenlong Wang, Charlie Nash, William L. Hamilton, David Duvenaud, Raquel Urtasun, and Richard Zemel. Efficient graph generation with graph recurrent attention networks. In *NeurIPS*, 2019.

[22] Jenny Liu, Aviral Kumar, Jimmy Ba, Jamie Kiros, and Kevin Swersky. Graph normalizing flows. *CoRR*, abs/1905.13177, 2019.

[23] Andrew Kachites McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. Automating the construction of internet portals with machine learning. *Inf. Retr.*, 3(2):127–163, July 2000.

[24] G. Mclachlan and K. Basford. *Mixture Models: Inference and Applications to Clustering*, volume 38. 01 1988.

[25] Luke O'Connor, Muriel M'edard, and Soheil Feizi. Maximum likelihood latent space embedding of logistic random dot product graphs. 2015.

[26] George Papamakarios. *Neural Density Estimation and Likelihood-free Inference*. PhD thesis, University of Edinburgh, 2019. Available at https://arxiv.org/abs/1910.13233.

[27] D.M.S.M. Penrose, M. Penrose, and Oxford University Press. *Random Geometric Graphs*. Oxford studies in probability. Oxford University Press, 2003.

[28] M. J. D. Powell. A hybrid method for nonlinear equations. In P. Rabinowitz, editor, *Numerical Methods for Nonlinear Algebraic Equations*. Gordon and Breach, 1970.

[29] Danilo Jimenez Rezende and Shakir Mohamed. Variational inference with normalizing flows. *ArXiv*, abs/1505.05770, 2015.

[30] Oleksandr Shchur and Stephan Günnemann. Overlapping community detection with graph neural networks. *CoRR*, abs/1909.12201, 2019.

[31] Johan Ugander, Brian Karrer, Lars Backstrom, and Cameron Marlow. The anatomy of the facebook social graph. *CoRR*, abs/1111.4503, 2011.

[32] Stanley Wasserman and Katherine Faust. *Social network analysis : methods and applications*. University Press, Cambridge, 1995.

[33] Duncan J. Watts and Steven H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440–442, June 1998.

[34] Jiaxuan You, Rex Ying, Xiang Ren, William L. Hamilton, and Jure Leskovec. Graphrnn: A deep generative model for graphs. *CoRR*, abs/1802.08773, 2018.

[35] Stephen J. Young and Edward R. Scheinerman. Random dot product graph models for social networks. In Anthony Bonato and Fan R. K. Chung, editors, *Algorithms and Models for the Web-Graph*, pages 138–149, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[36] Mingyuan Zhou. Infinite edge partition models for overlapping community detection and link prediction. 01 2015.